THE UNIVERSITY OF ALBERTA

RELEASE FORM

NAME OF AUTHOR John Cale Nesbit

TITLE OF THESIS Approximate String Matching Applied to Response Analysis in Computer Assisted Instruction

DEGREE FOR WHICH THESIS WAS PRESENTED Master of Education

YEAR THIS DEGREE GRANTED Fall 1983

THE UNIVERSITY OF ALBERTA

Approximate String Matching Applied to Response Analysis in Computer Assisted Instruction

by

©
John Cale Nesbit

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES AND RESEARCH

IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE

OF Master of Education

Department of Educational Psychology

EDMONTON, ALBERTA

Fall 1983

THE UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES AND RESEARCH

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research, for acceptance, a thesis entitled Approximate String Matching Applied to Response Analysis in Computer Assisted Instruction submitted by John Cale Nesbit in partial fulfilment of the requirements for the degree of Master of Education.

Abstract

Diverse algorithms have been proposed and implemented which allow machine recognition of the similarity or equivalence of two different character string representations of a single word. In CAI (Computer Assisted Instruction) these can be used to recognize student responses as alternate or incorrect spellings of a target word specified by a course author. Experiments were performed in order to compare the accuracy of some available approximate string matching functions and to develop an optimized function suitable for response analysis in CAI. A version of the edit distance algorithm, with edit costs for characters dependent on the probability of corruption of the character, was found to be superior for the sample of data used. The use of approximate string matching with response markup and dictionary features is discussed.

A rationale for writing a dissertation (besides the obvious one) should have something to do with the desire to shorten someone else's path to a given point so that he may press on to more interesting horizons.

Alan Kay, *The Reactive Engine*

Acknowledgements

The people who have shortened my path to this point include Tom Garraway, Don Lee, Dave Matheson, Norm McGinnis, the members of my committee, and the staff of the Division of Educational Research Services. I would particularly like to thank my supervisor Dr. Steve Hunka for his support, comments, and editorial tenacity.

Table of Contents

List of Tables

List of Figures

I. The Approximate String Matching Problem

In computer assisted instruction (CAI), the analysis of student responses requires a test which determines if a response belongs to a defined set. Consider the case where the student response is a string of characters representing an English word. The problem facing a course author using a conventional programming language is how to define, without resorting to the unpleasant method of enumeration, the set of character strings which he will accept as representing the word he is expecting. The set may include alternate spellings, incorrect spellings, differences in letter case, differences in tense, and possibly synonyms; all of which are equivalent for the purposes of the course author.

Specialized CAI authoring languages usually allow an author to define sets of strings by using some convenient and economical notation. For example, in Coursewriter II, an early CAI language (IBM, 1968), authors could use the notation *sp*l** to indicate a large set of acceptable strings such as *spell*, *spill*, *spwlx*, *spqlrst*, and *spelling* but excluding strings like *abcde*, *spaall*, and *spel*. The Coursewriter interpreter checked every ith character in the authors notation with the ith character in the student response. All characters but asterisks were required to match exactly. Asterisks in the final position could match with any substring, but other asterisks could match only one of any character.

Mentor, an early system supporting instructional dialogues (Feurzeig, 1969), recognized certain student responses as misspellings of a word given by an author. Feurzeig provides an excerpt from a dialogue teaching medical diagnosis where mentor accepts 'cuogh' as an alternative to 'cough'.

The success of any approximate string matching algorithm depends on how closely the set of responses acceptable to the algorithm corresponds to the set acceptable to the author. A more formal statement of this criterion requires the definition of a few terms. Consider a function f which accepts two arguments: t, a "target" string supplied by the author, and s, a string parsed from, or by itself constituting, a student's entered response. f returns a one if the two strings match or a zero otherwise. Over the lifetime of f in some CAI environment, t will take on n values or attributes which may be together represented as the vector T made up of t_1, t_2, ..., t_j, ..., t_n. For each member of T there will be m_j occurrences of s corresponding to the m_j occasions that f is invoked with the argument t_j. So, the values s takes on may be represented by the matrix S as in Figure 1. The total number of times f is invoked may be expressed as $\sum_{j=1}^{n} m_j$. The frequency that a one is returned will be:

$$\sum_{j=1}^{n} \sum_{i=1}^{m_j} f(s_{ij}, t_j)$$

Figure 1: Author and Student Strings

$$T = \begin{bmatrix} t_1, & t_2, & \ldots, & t_j, & \ldots, & t_n \end{bmatrix} \quad \text{author strings}$$

$$S = \begin{bmatrix} s_{11}, & s_{12}, & \ldots, & s_{1j}, & \ldots, & s_{1n} \\ s_{21}, & s_{22}, & \ldots, & s_{2j}, & \ldots, & s_{2n} \\ \vdots & \vdots & & \vdots & & \vdots \\ s_{i1}, & s_{i2}, & \ldots, & s_{ij}, & \ldots, & s_{in} \\ \vdots & \vdots & & \vdots & & \vdots \\ s_{m_1 1}, & s_{m_2 2}, & \ldots, & s_{m_j j}, & \ldots, & s_{m_n n} \end{bmatrix} \quad \text{student strings}$$

The frequency that a zero is returned will be:

$$\sum_{j=1}^{n} m_j - \sum_{j=1}^{n} \sum_{i=1}^{m_j} f(s_{ij}, t_j) \quad .$$

If an author, given t_j and s_{ij} were able to personally perform all $\sum_{j=1}^{n} m_j$ judgements, one might say that his behavior defined a function called f'.[1] The frequency of agreement and disagreement between f and f' is presented in Figure 2 as a 2x2 table. When f returns a one and f' returns a zero, f is making a type I error the frequency of which is given in the lower left quadrant. When f returns a zero and f' returns a one, f is making a type II error, the frequency of which is given in the upper right quadrant.

----

[1] As in Thorelli's (1962) description of "man as a secondary machine" in relation to the problem of error correction in text.

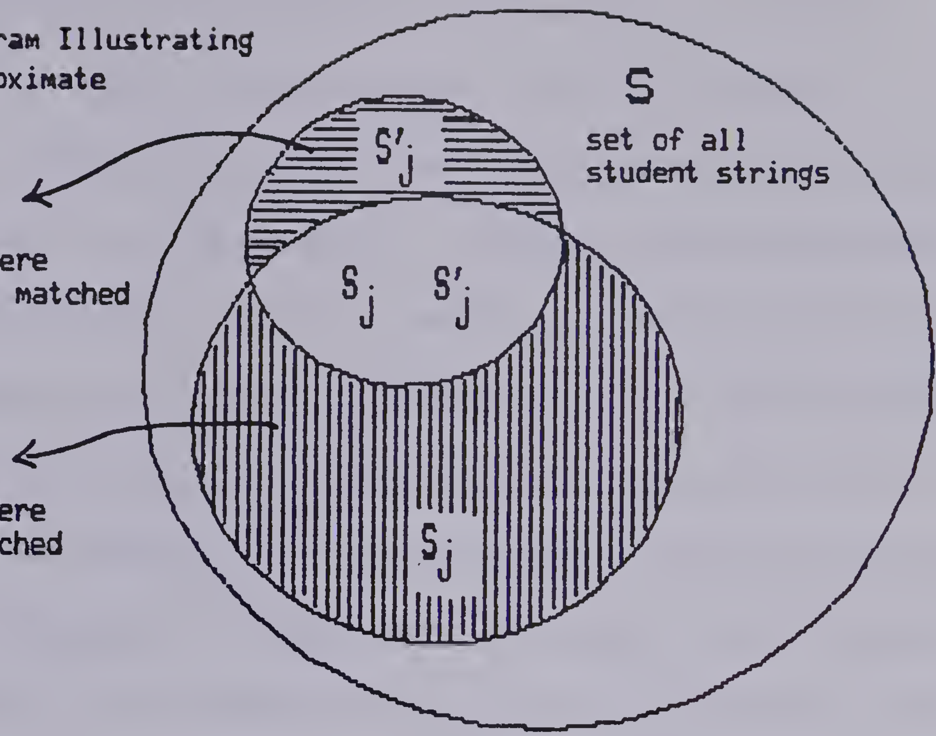
Figure 2: Error Frequencies

		Author's decision (f') 1	0	
Machine's decision (f)	1	agreement $\sum_{j=1}^{n}\sum_{i=1}^{m}(f(s_{ij},t_j)f'(s_{ij},t_j))$	type II error $\sum_{j=1}^{n}\sum_{i=1}^{m}(f(s_{ij},t_j)\lvert f'(s_{ij},t_j)-1\rvert)$	$\sum_{j=1}^{n}\sum_{i=1}^{m}f(s_{ij},t_j)$
	0	type I error $\sum_{j=1}^{n}\sum_{i=1}^{m}(\lvert f(s_{ij},t_j)-1\rvert f'(s_{ij},t_j))$	agreement $\sum_{j=1}^{n}\sum_{i=1}^{m}(\lvert f(s_{ij},t_j)-1\rvert\lvert f'(s_{ij},t_j)-1\rvert)$	$\sum_{j=1}^{n}m_j-\sum_{j=1}^{n}\sum_{i=1}^{m}f(s_{ij},t_j)$
		$\sum_{j=1}^{n}\sum_{i=1}^{m}f'(s_{ij},t_j)$	$\sum_{j=1}^{n}m_j-\sum_{j=1}^{n}\sum_{i=1}^{m}f'(s_{ij},t_j)$	$\sum_{j=1}^{n}m_j$

One might also present the problem by means of a venn diagram (Figure 3) where the outer circle S represents the set of all student responses. For every element of T, f defines a subset of S called S_j consisting of those responses which cause f to return a one. Similarily, for each element of T, f' will define a subset S'_j. Therefore, $S_j-(S_j \Omega S'_j)$ represents the student strings resulting in type I error and $S'_j-(S_j \Omega S'_j)$ represents those producing a type II error for the jth member of T.

The problem of designing an optimal approximate string matching function for a particular CAI application is the problem of maximizing the agreement between that function, f, and the author's judgement, f', while minimizing some appropriate balance of type I and type II errors. The purpose of this study was to determine which algorithms most

Figure 3: Venn Diagram Illustrating the Problem of Approximate String Matching



effectively resolve this problem and are thus most deserving of inclusion in CAI languages and response analysis systems.

Chapter I is a review of some approximate string matching algorithms found in CAI and other applications. Chapter II reports on experiments comparing some of the algorithms reviewed, and evaluations of modifications introduced by the present author. In Chapter III, consideration is given to applying the findings of Chapter II to the design of CAI languages and response analysis systems. Related problems such as the determination of equivalence of algebraic statements or of English sentences are not within the scope of this thesis.

In their review of string matching techniques applied to information retrieval from data bases, Hall and Dowling (1980) established a number of definitions and conventions

which are followed here wherever they are relevant to response analysis. Of notable importance, is their distinction between **string similarity** and **string equivalence**. These and other definitions are cast into a form applicable to the general problem outlined above.

Given strings t and s, a string similarity function returns a value r_{st} which represents the proximity of t and s. The definition of proximity is determined by the particular similarity function at hand. Similarity functions are characterized by reflexivity ($r_{st}=r_{st}$) and symmetry ($r_{st}=r_{ts}$), but not transitivity. It is frequently useful to force a binary result on a similarity function by setting a threshold R on r_{st} to produce a new similarity relation r'_{st}. In this case r'_{st} indicates whether s is a member of S_j which includes and is specified by t_j.

Equivalence functions are a subclass of similarity functions which return a binary result and are transitive (where x is a third string, if $r_{st}=r_{tx}$ then $r_{st}=r_{sx}$). According to Hall & Dowling, this implies that:

> The equivalence relation divides the set S of all strings into subsets S_1, S_2, S_3, ... such that all strings in a subset are equivalent to each other and not equivalent to any string in any other subset. These subsets are called "equivalence classes".

Therefore, when f is an equivalence function, S_j is the **equivalence class** determined by t_j.

There exists a superclass of similarity functions, **approximate string matching functions**, whose members are constrained by neither transitivity nor symmetry. The

matching facility in Coursewriter illustrated above falls only into this common class.

String Equivalence Functions

The Soundex method (Table 1), hereafter referred to as SOUNDEX, qualifies as the earliest and most widely known string equivalence function. The algorithm given in Table 1 is apparently the product of modifications made at the Remington-Rand corporation to a system for filing documents originally patented in 1918 by R.C. Russell. In Russell's system[2] a code for a name was created by subjecting all characters except that in the initial position to a series of simple transformations. All instances of *h*, *w*, final *s*, final *z*, or the digraph *gh* were deleted. The remaining characters were replaced by a digit which grouped together letters representing similar sounds as in Table 1. Finally, identical adjacent digits and all but the initial instance of the digit representing vowels were deleted. Later modifications introduced truncation to four characters and deletion of all instances of the vowel digit, but abandoned the initial deletion of final *s*, final *z*, and *gh*.

Hall and Dowling refer to the SOUNDEX code, and comparable codes in other equivalence functions, as **canonical forms** which define an equivalence class. The canonical form may be thought of as the minimally informative string from which all members of the equivalence

---

[2] Two patents were registered with the U.S. patent office, number 1261167 in 1918 and number 1435663 in 1922.

Table 1: The SOUNDEX Algorithm

Do the following operations on t and s independently to generate their canonical forms:

1. Set the first chararacter in the canonical form to be the first letter in the original string.
2. Use this chart to replace the letters in the original string with the corresponding digits.

0	*a, e, i, o, u, h, w, y*
1	*b, f, p, v*
2	*c, g, j, k, q, s, x, z*
3	*d, t*
4	*l*
5	*m, n*
6	*r*

3. Delete all zeros.
4. Delete all repeated adjacent digits.
5. Truncate to four characters.

If t and s have the same canonical form then they are equivalent; otherwise they are not equivalent.

Examples:			Counterexamples:		
machine	⇒	M25	*bridge*	⇒	B623
masheen	⇒	M25	*brige*	⇒	B62
filament	⇒	F453	*decision*	⇒	D25
fixture	⇒	F236	*disown*	⇒	D25

class it represents can be generated. Although any member of the equivalence class can generate all the other members, all except the canonical form have information which is redundent to that task. The loss of information in the derivation of the canonical form is manifested as a reduction in string length, a reduction in alphabet size, or both. In the case of SOUNDEX, there is shrinkage in both the alphabet size (26 letters to 6 digits in all but the first character) and in the length (resulting from truncation and deletion of characters).

Refering to Figure 3, one might restate the problem of getting S_j to approach S'_j as that of inventing rules for the derivation of a canonical form such that only information which discriminates between the elements of S'_j is discarded. One characteristic which the elements of S'_j are likely to share in common is their phonetic interpretation. Masters (1927) concluded that in his sample of 13,183 misspellings of 278 words, 65% were "possible spellings from a phonetic point of view". This suggests that, where S'_j consists of a word and its misspellings, a function is needed which generates canonical forms which are representations of the way words sound. This is the rationale behind many of the functions discussed here.

Among the misspellings which can confound matching functions relying on phonetic analysis techniques are those resulting from incorrect or alternate pronunciations. Masters identified an additional 14% of his sample as follows:

> Misspellings which, though they cannot be pronounced exactly like the correct form, are approximate phonetic spellings; which are possible phonetic spellings for common mispronounciations of the words; or in which the necessary change in the pronounciation of this form is so slightly different from the exact pronounciation of the correct form that it is scarcely detectable by the uncritical ear.

A matching function based on an algorithm which generates a precise phonetic representation from an orthographic representation may fail to recognize misspellings falling into this latter category.

Hewes and Stowe (1965) point out that SOUNDEX attempts to solve this problem by grouping together letters commonly representing frequently confused phonemes under the same code element. The bilabial stops (/P/ and /B/) are thrown together with the labiodental fricatives (/F/ and /V/).[3] Since *h* is deleted, the alveolar stops (/T/ and /D/) exist in the same group as the dental fricatives (the initial consonants in *thigh* and *thy*). The often confused nasal resonants (/M/, /N/, and /ING/) are identified by a single code.

Because they are rarely confused by native English speakers, the lateral and median alveolar resonants (/L/ and /R/) were allowed separate code elements. The velar stops (/K/ and /G/) are combined with the affricated alveopalatal stops (/CH/ and /J/) and the groove fricatives (/S/, /Z/, /SH/, and the first consonant in *azure*) because English orthography fails to preserve any simple consistent distinction between these phonemes.

Although the notorious variability of vowel pronounciation may justify the mapping of all vowels to a single code element, it would not seem to support the elimination of that element from the final canonical form. One can probably assume that the characters *Y*, *H*, and *W* are mapped to the vowel code element because they rarely signal

----

[3] Phonemes are here indicated by the lexeme (in upper case) with which the phoneme is usually associated, enclosed by slashes. In cases where this *ad hoc* system fails, meaning has been clarified by example. See Gleason (1961) for an explanation of phoneme nomenclature.

a consonant except in the initial position -- which is not transformed in any case.

If the vowel code element is to be deleted, it seems unreasonable to do so before the elimination of repeated adjacent code elements. Hewes and Stowe presented a method which is identical to that given in Table 1 except that steps 3 and 4 are interchanged. The modified version preserves the disjunction of identical code elements separated only by vowels and will thus maintain a distinction between words like "phases" and "packages".

The PHONETIC option for response analysis in the CAI language PLANIT (Feingold, 1966; Butler and Frye, 1970) is illustrated in Table 2. The PLANIT algorithm, hereafter referred to as PLANIT, is based on the modified soundex method of Hewes and Stowe with two notable differences:

1. The initial character is not exempted from the transformation procedure, but the transformed character in this position is never deleted.
2. The 'H' and 'W' characters are mapped to a single code element separate from the vowel code element.

One justification implied by Hewes and Stowe for the preservation of the initial character is that, where the SOUNDEX code is serving as a key in an information retrieval system, each of the 26 alphabetic characters can indicate an absolute address from which a sequential search for a matching code can proceed. Perhaps a more significant argument for the preservation of the initial character is

that the frequency of misspellings in the initial position is much lower than in the rest of the word.

If the initial character is transformed, it follows that *h*, *y* and *w* should no longer be mapped to the vowel code element. In the initial position of certain words of Old English origin, the bilabial median resonant preceded by breath (/HW/ as in *wheat*) was actually represented by the digraph *hw* well into the 13th century.[4] After the introduction of *wh* to represent this vocalization, the orthography was further obscured by the disappearance of the preceding /H/ in most modern dialects, and by the adoption of *wh* to represent /H/ in the initial position of words such as *whom*. This is apparently the reason in PLANIT for grouping *h* and *w* together under the same code element. There seems to be no justification for leaving *y* in the vowel group rather than allowing a separate code element which is deleted in all but the initial position.

One can recognize two fundamental weaknesses in SOUNDEX and its descendents: the inability to parse lexemes containing more than one character, and the inflexible way of handling ambiguity which requires a lexeme to be mapped to only one code element. Although these limitations allow for relatively quick execution, it is debatable whether modern CAI systems can afford the inaccuracy they impose.

The identification of common digraphs and trigraphs through a parsing mechanism would remedy many of the

---

[4] *Oxford English Dictionary*. London: Oxford University Press.

Table 2: The PLANIT Algorithm

Do the following operations on t and s independently to generate their canonical forms:

1. Use this chart to replace the letters in the original string with the corresponding letters:

A	*a, e, i, o, u, y*
B	*b, f, p, v*
C	*c, g, j, k, q, s, x, z*
D	*d, t*
H	*h, w*
L	*l*
M	*m, n*
R	*r*

2. Except for the first character, delete all occurrences of H.
3. Delete all repeated adjacent characters.
4. Delete all occurrences of A.

Examples:	Counterexamples:
machine ⇒ MCM	*bridge* ⇒ BRDC
masheen ⇒ MCM	*brige* ⇒ BRC
disown ⇒ DCM	*acid* ⇒ CD
decision ⇒ DCCM	*quiet* ⇒ CD

problems arising with the soundex method. For example: *ng* can be identified correctly as /ING/; and *dg* can be recognized as /J/, allowing a successful match of the names *Rodgers* and *Rogers*. However, a problem would still be posed by lexemes like *t*, which commonly represents /T/ (as in *negative*) but may also represent /SH/ (as in *negotiate*). Some method is clearly needed which allows several different lexemes to be associated with a common string of phonemes and conversely, several different phoneme strings to be associated with a common lexeme.

Symonds (1970) proposed an algorithm (Table 3), hereafter referred to as SYMONDS, which later formed the basis for the cp (compare phonetic) function in the National Research Council's NATAL-74 (Westrom,1974) and Honeywell's NATAL II (Honeywell,1981) authoring language. Instead of generating a single canonical form, SYMONDS multiplied the current number of canonical forms by the number of alternate phonemes associated with the current lexeme in the string. This resulted in $\prod_{i=1}^{n} q_i$ canonical forms where q is the number of alternate phonemes corresponding to the lexemes parsed from the string and n is the number of these lexemes. If two strings shared at least one canonical form, they were considered equivalent. Although Symond's method can be viewed as an extrapolation from the simpler soundex-type methods, the generation of multiple canonical forms destroys the mutual exclusivity of equivalence classes so, strictly speaking, the method cannot be considered an equivalence function.

SYMONDS appears to be free from many of the faults and restrictions of the soundex-like functions; albeit at the cost of significantly greater execution time. However, several details of the mapping algorithm are questionable. No example is given justifying the phoneme /G/ as an alternate for the digraph *dg*. One would think the digraph should be mapped down to /J/ and /DG/ (as in *Edgar*). Similarily, no example is given for the phoneme /SH/ as a translation of *sce* or the phoneme /K/ as a translation of

Table 3: The SYMONDS Algorithm

Use the chart to independently build a series of canonical forms for each of s and t under the following constraints:

1. Vowel characters a,e,i,o,u are skipped over and are not represented in the canonical form (except the i in sci and the e in sce).
2. Adjacent identical consonants are skipped.
3. y is skipped over unless it appears in the first or last position.
4. w is ignored if it appears in the last position.
5. When two lexemes both match to the beginning of the unparsed string, use the lexeme with the most characters.
6. When the lexeme has more than one corresponding phoneme representation, produce enough copies of the current set of canonical forms so that each new phoneme representation can be appended to a copy of each form in the current set. All of the unique forms so produced now become the new current set.

If s and t have any canonical forms in common, they are considered equivalent.

lexeme	phoneme		lexeme	phoneme	
b	B	but	p	P	pen
ch	s	machine	q	Q	queen
	C	chair			
ck	K	back	r	R	rat
cq	Q	acquire	rh	R	rhubarb
c	S	city	sci	S	science
	K	can		s	conscience
dg	J	badge	sce	S	scene
	G			s	
d	D	dog	sh	s	she
f	F	fill	s	S	safe
gn	N	gnaw		Z	easy
ght	T	slaughter	tch	s	
	FT	laughter		C	latch
gh	F	laugh		K	
	G	ghost	tio	C	question
g	J	gem		s	nation
	G	gum	th	t	the
h	H	hat	t	T	take
j	J	joke	v	V	van
kn	N	knot	wh	W	when
k	K	keep	w	W	way
l	L	late	x	KS	vex
m	M	man		GZ	exist
n	N	nod	y	Y	yet
ph	F	phantom	z	Z	zoo

Examples:	Counterexamples:
bridge ⇒ BRJ,BRD	quay ⇒ QY
brige ⇒ BRJ,BRG	kway ⇒ KWY
acid ⇒ SD,KD	cite ⇒ KT,ST,CT
quiet ⇒ QT	kite ⇒ KT

tch. The existence of the 'phoneme' Q seems particularily unreasonable because a mapping to /KW/ (as in *queen*) and /K/ (as in *daqueri*) would serve better.

One major deficiency is the absence of a general mechanism for incorporating positional information about a lexeme into the mapping procedure. As with the digraph *gh*, which never represents /F/ in the initial position and never represents /G/ in the final position, knowing the position of a lexeme can frequently remove ambiguity from its translation.

SYMONDS is incapable of identifying most cases of misspellings resulting from incorrect or alternate pronunciation (the approximate phonetic misspellings described by Masters). Misspellings such as *dem* for *them* and *ting* for *thing* will not be recognized.

Problems relating to the retrieval and storage of information have motivated most of the work on approximate string matching algorithms. The extent to which misspellings can hamper the searching of data bases was documented by Bourne (1977) who found the frequency of index term misspellings in a sample of 11 bibliographic data bases to

range from 0.5% in one data base to almost 23% in another. While the recognition of equivalent or similar keywords has been the goal most relevant to response analysis in CAI, early work on the abbreviation of stored English words in an era of expensive memory deserves some mention here because it can be viewed as a related problem involving the generation of canonical forms.

Bourne and Ford (1961) compared several methods for the abbreviation of English words and names according to the dual criteria of compactness and discriminability of the abbreviated form. The methods considered by Bourne and Ford ranged from truncation of the string (from either end) to procedures where some mathematical function is applied to the internal numeric representation of the string. Most of the methods generated a canonical form by eliminating characters selected according to a set of simple rules. Only a few of the methods described are reasonable approaches to the approximate string matching problem. These are given in Table 4.

To apply the last three methods to the word recognition problem, rather than a ranking of frequency of usage, one requires a ranking of the frequency of misusage of characters. A method for the recognition of misspellings proposed by Blair (1960, Table 5), hereafter referred to as BLAIR, bears some similarity to the third algorithm in Table 4, but was modified to use an error frequency ranking. Blair stated that his coding algorithm based on error frequency is

Table 4: Bourne & Ford's Abbreviation Algorithms

1. **Elimination of Vowels**
   Starting from the right end of the string delete the characters a,e,i,o,u until the required string length is achieved or the left end of the string is reached. In the latter case, truncate the remaining string to the required length.
2. **Elimination by Character Frequency**
   Given a ranking of all alphabetic characters by frequency of usage obtained from a sample of words similar to those one expects to operate on, delete the most common characters until the required length is achieved. For words the ranking was:
   EIROATNSLCPMDUHGYBFVKWXZJQ.
   For personal names the ranking was:
   EARNLOISTHDMCBGUWYJKPFVZXQ.
3. **Elimination by Positional Character Frequency**
   Given a separate frequency ranking of all alphabetic characters for each character position, keep finding and eliminating the highest ranking character until the required length is achieved. In the case of ties, delete the rightmost character first. It was reported that the differences between the rankings for each position were very small for all but the first three positions, which each differed markedly from the other rankings.
4. **Elimination by Character Bigram Frequency**
   Given a frequency ranking of bigrams, determine a score for each character by summing the ranks for the two bigrams to which the character belongs. Then start eliminating the characters with the highest scores until the required length is achieved. Bigrams may include spaces, but the initial character is retained regardless of its score.

superior to one using simple character frequency, but unfortunately no details of this comparison were supplied.

Although the nominal and position scores were reported to be based on the "frequency of their occurrence as errors", there is no explicit description of how they were

derived. But taking the quote literally it is apparent that there was a failure to include the frequency of nonoccurrence of correct characters in misspellings. This is important because not only misspellings but also the correct words are being reduced to canonical form. One possible way of generating such information would be to correct a large sample of misspelled words using only operations of insertion and deletion. With each operation a score associated with the character serving as the operand would be incremented. It appears that Blair's scores would be comparable to those derived were one to only count the deletions but not the insertions.

Davidson (1962) used a coding method which was successfully employed in the retrieval of passenger names from an airline's record system. It reduced each passenger's name to a five character code by means of vowel deletion, deletion of repeated adjacent characters, and truncation. The inclusion of the first initial as the fifth character of the code raises the interesting possibility of special coding techniques for certain classes of words.

One expects classes of words or phrases such as personal names, addresses, adjectives, various forms of specialized jargon, and so on, to have different equivalence relations. If *Lawrence Street* refers to a place, then *Lawrence St.* is equivalent but *L. Street* probably is not. But the converse is true if a person is being referenced. If separate approximate string matching functions for

Table 5: The BLAIR Algorithm

Use the following paired list to obtain a nominal score for each letter in the word:

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
5	1	5	0	1	1	2	5	6	0	1	5	1	3	4	3	0	4	5	3	4	1	1	0	2	1

Next use the following paired lists to obtain a positional score for each letter in the word. Select a score for the first character from list #1 then select a score for the last character from list #2 then satisfy the second character from #1 and the penultimate character from #2 and so on until every character has been assigned one score:

#1

position	1	2	3	4	5	6	7	8	9	10...
score	0	2	4	5	5	6	6	6	7	7...

#2

position	1	2	3	4	5	6	7	8	9	10...
score	1	3	4	5	5	6	6	7	7	7...

Finally, find the deletion rating of each letter by taking the sum of its nominal and positional scores and then delete the letters with the n highest deletion ratings.

distinguishable vocabularies are shown to sufficiently optimize the recognition process, it may be desirable to provide the CAI author with a toolkit of specialized functions. A further step would be to enable the author to construct a function to suit the job at hand, either from scratch or by modifying existing functions.

In his monograph *Information Retrieval and the Computer*, Paice (1977) considered the recognition of abbreviations, synonyms, alternate correct spellings, and word roots. Of course, the absence of rules relating different strings with similar meanings requires methods for the recognition of synonyms to rely on something like a

thesaurus. The prospects for the recognition of abbreviations and contractions are not much better. Paice presented four rules for the generation of a canonical form useful in the recognition of alternate correct spellings (Table 6). A rule which deleted consonants occurring internally would probably be an effective addition to this set since British spellings frequently have these where American spellings do not.

A CAI author may be willing to accept a word from the student response having the same root as his target word. Paice's "conflation" algorithm (Table 7) attempts to remove suffixes. Paice points out that finding the correct root is not necessary as long as "(i) members of a family reduce to the same root, and (ii) members of different families reduce to different roots". He also notes that procedures for removing prefixes are less useful because the result of such an operation is frequently a word with an opposite or radically different meaning. The algorithm in Table 7 is executed by beginning at the label START and checking the ending given in the table against the ending of the string being conflated. If the ending matches, replace it with the replacement given and follow the corresponding transfer instruction. Otherwise, step to the next line and reiterate the procedure. The dash represents that substring which encompasses all characters excluding the suffix.

The applicability of the methods considered here to response analysis in languages other than English will

Table 6: Paice's Alternate Spelling Conversion Rules

(¬ represents some substring)

1. Change *z* to *s* using the rule

   ¬VzV¬ => ¬VsV¬

   where V is a,e,i,o,u or y.
   Examples: *razor, analyze, realize*
   Counterexamples: *hazard, squeeze*

2. Change *ph* to *f* using the rule

   ¬ph¬ => ¬f¬

   Examples: *sulpher, peripheral, symphony*
   Counterexamples: *uphill, haphazard*

3. If the original word has a length > 5 then apply the rule

   ¬our => ¬or

   Examples: *flavour, humour, four*
   Counterexample: *devour*

4. After removing endings such as *-e, -ate, -ation,* apply the rule

   ¬tr => ¬ter

   Examples: *centr(e), filtr(ate), titr(ation)*

probably vary with the similarity of the language to English. Fendt (1974) documented the use of the PLANIT algorithm in a German CAI system consisting of a collection of APL functions. On the other hand, in languages with a systematic orthography, such as the Japanese hiragana and katakana, the need for approximate string matching functions may be negligible.

One way of summarizing and classifying the equivalence functions reviewed is by viewing each edit operation contributing to the conversion of a word to a canonical form, as a substitution operation which replaces a substring of length m with a substring of length n. When m=0 and n>0, an insertion occurs. When m>0 and n=0, a deletion occurs.

Table 7: Paice's Conflation Algorithm

label	ending	replacement	transfer
START	*-ably*	-	goto IS
	-ibly	-	stop
	-ly	-	goto SS
SS	*-ss*	*-ss*	stop
	-ous	-	stop
	-ies	*-y*	goto ARY
	-s	-	goto E
	-ied	*-y*	goto ARY
	-ed	-	goto ABLE
	-ing	-	goto ABLE
E	*-e*	-	goto ABLE
	-al	-	goto ION
ION	*-ion*	-	goto AT
	-	-	stop
ARY	*-ary*	-	stop
	-ability	-	goto IS
	-ibility	-	stop
	-ity	-	goto IV
	-ify	-	stop
	-	-	stop
ABL	*-abl*	-	goto IS
	-ibl	-	stop
IV	*-iv*	-	goto AT
AT	*-at*	-	goto IS.
IS	*-is*	-	stop
	-ific	-	stop
	-olv	*-olut*	stop
	-	-	stop

The simplest coding procedures represented by the methods of Davidson, Blair, and those of Bourne and Ford given in Table 4, perform only single character deletion (m=1,n=0). Two simple rules seem to pervade these and other methods using character deletion: *delete vowels* and *delete adjacent repeated characters*. While the latter rule is absent from the the coding procedures of Blair, and Bourne and Ford, the effect of the *delete vowels* rule is achieved

by assigning most vowels high weights to increase the probability of their deletion.

The conclusion that these two rules together constitute a tolerably successful method of generating canonical forms is supported by the experience of workers at the University of Alberta. One extension to Coursewriter II (Romaniuk and Schienbein, 1973) developed by Peuchot, which also appeared later in his IMOGENE instruction module generator (Peuchot,1975), allowed authors to enter a 'skeleton' of a student response they were expecting. The skeleton word was actually a canonical form which successfully matched with any word containing the characters of the skeleton in correct order but ignoring absolute character position. Authors usually generated the skeleton by simply removing all vowels and repeated consecutive consonants.

Slightly more complex equivalence functions which allow substitution with m=1 and n=1 are represented by the soundex-like methods. For these methods, an attractive alternative to deleting the vowels altogether is to map them all to a single character before applying the *delete adjacent repeated characters* rule. If words tend to be misspelled by the substitution of vowels for other vowels, then the reduction in type II errors would outweigh the increase in type I errors. Under the proposed rule, *brake* would not be confused with *bark*. However, if words tend to be misspelled by the deletion of vowels, then the increase in type I error would probably be too high. For example,

brak would not be judged equivalent to *brake*. This problem might be obviated by deleting any final *e* before transforming the vowels.

The most complex transformation procedures according to this classification scheme allow substitution with values of n and m exceeding one. The functions falling into this category, SYMONDS and Paice's alternate spelling and conflation routines, all require a parsing of the original string.

String Similarity Functions

Recall that, given strings t and s, a similarity function returns a value r_{ts} which is a measure of the proximity of t and s. Hall and Dowling noted that the similarity relation can be used either to find all strings $t_1, t_2, t_3, \ldots t_n$ such that r_{ts} is "greater than some threshold of acceptability" (previously called R), or to find "the N strings $t_1, t_2, t_3 \ldots t_n$ such that their $[r_{ts}]$ have the N largest values". These formulations reflect procedures associated with the retrieval of information where an index term entered by a user is compared against index terms linked with the stored data. Although similar procedures may be used in an instructional program to query the student about the intended response by displaying all target words meeting some threshold, consideration is given here only to the comparison of s and t such that a binary decision r_{ts} is returned.

Hall and Dowling observed that while the range of r is arbitrary, "the value of +1.0 for an exact match seems to have strong intuitive appeal, and the range of values from -1.0 to +1.0 appears to gain respectability from correlation coefficients and normalized inner products". The range 0.0 to 1.0 has been more commonly used perhaps because the similarity between two strings, by analogy with physical distance, is heuristically an unsigned magnitude for most functions.

The similarity relation can be viewed as more appropriate than the equivalence relation for the problem of approximately matching English words due to the intolerance of the latter to ambiguity. Mutually exclusive equivalence classes cannot represent the frequent occasion where a feature of the word is similar to two or more other features which are themselves dissimilar. For example, in an orthographic sense, *c* is similar to both *s* and *k*; but *s* and *k* are dissimilar. This was seen to be the case with SYMONDS, which needed to destroy the mutual exclusivity of equivalence classes in order to accurately interpret the relationship between lexemes.

The earliest publication describing the string similarity relation is apparently that of Glantz (1957). He proposed a function which simply padded the shorter string with blanks and tested the two characters at each position in the strings for equality. The number of mismatched characters was divided by the string length to yield a value

of r which ranged from 0.0 to 1.0.

Damerau (1964) observed that more than 80% of keypunching errors were single instances of either insertion, deletion, substitution, or adjacent transposition of characters in a word. He suggested that "these are the errors one would expect as a result of misreading, hitting a key twice, or letting the eye move faster than the hand". An algorithm (Table 8) was proposed whereby two strings (A and B) are judged to be similar if one could be converted to the other by any one of these operations. DAMERAU has been used successfully to detect misspelling of keywords in compilers (Morgan, 1970) and operating systems such as MTS[5].

Faulk (1964) defined three measures of string similarity: material, ordinal, and positional. All three functions, and also a superordinate function which contains them, return values ranging from 0.0 to 1.0. Given the strings A and B, having lengths m and n, a *matching pair* of characters is represented by the coordinates (i,j). K is the complete set of d matching pairs between A and B. q is the number of pairings of matching pairs in which both the i and j coordinates of one matching pair are greater than those of the other matching pair. q has a maximum value of d^2-d. To illustrate, for the strings *abcd* and *adadec*, K becomes (1,1),(4,2),(1,3),(4,4),(3,6) and n=4, m=6, d=5, q=10. However, for comparisons involving "redundent" strings, those which contain more than one instance of each character

----

[5] Michigan Terminal System

Table 8: The DAMERAU Algorithm In Pseudo-Pascal

```
var    m,n,i,diff : integer;
       errorcount,firsterror,lasterror : integer;

procedure match(length);
    begin
    errorcount := 0;
    firsterror := 0;
    lasterror := 0;
    for i:=1 to length do
         begin
         if A[i] ≠ B[i] then
              begin
              errorcount := errorcount + 1;
              if errorcount = 1 then firsterror := i
              else lasterror := i;
              end;
         end;
    end;

begin
m := length(A);
n := length(B);
diff := m-n;
if (diff < -1) or (diff > 1) then print "dissimilar"
else case diff of
0:  begin
    match(m);
    if errorcount < 3 then
        case errorcount of
         0,1: print "similar";
           2: if (firsterror = lasterror-1)
              and (A[firsterror] = B[lasterror])
              and (B[firsterror] = A[lasterror])
              then print "similar"
              else print "dissimilar";
        endcase
    else print "dissimilar";
    end;
-1: begin
    match(m);
    if errorcount = 0 then print "similar"
    else begin
         delete(B[firsterror]);
         match(m);
         if errorcount = 0 then print "similar"
         else print "dissimilar";
         end;
    end;
 1: begin
    match(n);
    if errorcount = 0 then print "similar"
```

```
    else begin
         delete(A[firsterror]);
         match(m);
         if errorcount = 0 then print "similar"
         else print "dissimilar";
         end;
    end;
endcase;
end.
```

---

as is the case with the above example, K is not useful. Instead, Faulk defined K' to be the set of optimal matched pairs. That is to say, given that only a one-to-one correspondence of elements is allowed, the matched pairs (i,j) are chosen such that the sum of $(i-j)^2$ over all d matched pairs is minimal. With this definition, d can attain a maximal value of MIN(m,n). The example comparison above produces K' = (1,1),(3,6),(4,4) and d=3, q=4.

Now the three measures of similarity can be given as follows:

1. Material similarity = 2d/(m+n).
2. Ordinal similarity = $q/((m^2-m)/2 + (n^2-n)/2)$.
3. Positional similarity = $2(rmax-r)/((m^3-m)/3 + (n^3-n)/3)$.

   Letting X be an index to the elements of K, r is the total amount of *positional disparity*

   $$r = \sum_{x=0}^{d} (i_x - j_x)^2$$

   and rmax is the maximum value r can attain for a given value of d assuming nonredundant strings:

   $$rmax = \sum_{x=0}^{d} (MAX(m,n) - 2X - 1)^2$$

When d=m=n, rmax attains a maximum value of $(m^3-m)/3$ which is the basis for the normalizing divisor.

Finally a Total Similarity Function is given as:

(material + ordinal + positional)/3

which raises the question of optimal weightings for the subordinate functions.

Alberga (1967) reported on and tested approximate string matching functions developed by himself and others at the IBM Watson Research Center. Alberga chose to express the algorithms he presented as operations on a binary coincidence matrix E generated from two strings. E has the order m x n corresponding to the lengths of the two strings. E_{ij} is either 1 or 0 depending on the equality of the ith character in the first string with the jth character in the second string.

A general approach was proposed which decomposed the algorithms into three phases of operations on coincidence matrices: weighting, selection, and similarity operations. Several complete similarity functions (Alberga tested 65) can be constructed by choosing one operation for each phase (Table 9). The weighting and selection phases are optional.

Weighting operations assign to each element of the matrix a value related to the probability that one of the corresponding characters is derived from the other. ROOF is based on the principle that the matrix elements of derived pairs tend to cluster around the axis. The axis is defined as the 0th diagonal where the Kth diagonal was any subset of

Figure 4: Matrix Augmentation in CONTEX

$$
\begin{array}{ccccccc}
1 & 1 & 0 & 0 & \ldots\ 0 & 0 & 0 \\
1 & 1 & 0 & 0 & \ldots\ 0 & 0 & 0 \\
0 & 0 & E_{1,1} & E_{1,2} & \cdots\ E_{1,n} & 0 & 0 \\
0 & 0 & E_{2,1} & E_{2,2} & \cdots\ E_{2,n} & 0 & 0 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
0 & 0 & E_{m,1} & E_{m,2} & \cdots\ E_{m,n} & 0 & 0 \\
0 & 0 & 0 & 0 & \ldots\ 0 & 1 & 1 \\
0 & 0 & 0 & 0 & \ldots\ 0 & 1 & 1
\end{array}
$$

elements such that i-j equaled some constant K.

CONTEX is based on the reasonable notion that the probability that a character match is not spurious is dependent on whether the surrounding characters match as well. The augmentation of the matrix is the same as adding two delimiters to both ends of the string which match each other but none of the internal characters. The weighting factor for CONTEX given in Table 9 is a *post hoc* replacement suggested by Alberga for an apparently erroneous factor he used in his comparative study.

The result of the selection phase, is generally an m x n matrix having no more than one nonzero element in each row and column. Note that the SFIRST, SORDER, and LSTNG operations are unaffected by any weighting operations which precede them. The asymmetry which characterizes SFIRST,

Table 9: Alberga's Matching Operations

Weighting Phase

1. ROOF
   Multiply every i,jth element by $1-D_{ij}$ where D_{ij} is the distance between that element and the axis:
   $$D_{ij}=|(i-1)/(m-1)-(j-1)/(n-1)|$$
2. CONTEX
   Augment the matrix by adding four rows and four columns as in Figure 4. Apply the following weighting factor to each element E_{ij}:
   $1/14(1+3(E_{i+1,j+1}+E_{i-1,j-1})+$
   $2(E_{i+2,j+2}+E_{i-2,j-2})+E_{i+1,j+1}E_{i+2,j+2}+$
   $E_{i-1,j-1}E_{i-2,j-2}+E_{i+1,j+1}E_{i-1,j-1})$

Selection Phase

1. SFIRST
   Starting with the top row, search each row from left to right and find the first nonzero element E_{ij}. Zero all the other elements in that row. Start the search in the next row at the (j+1)st column. Halt the operation and zero any remaining rows when either i=m or j=n.
2. SORDER
   This operation is the same as SFIRST except that when no nonzero elements can be found in a row, the search is resumed at the (i+2,j+2)th element.
3. SBYC
   Starting at the top row, find the largest element in each row not occupying a columnar position held by a previously selected element. Zero all other elements in that row.
4. SMAX
   Find the largest element in the matrix. Zero all other elements occupying the same row and column. Continue this procedure by finding the next largest elements until all rows and columns have been processed.
5. LSTNG
   Examine each diagonal to find the largest set of consecutive nonzero elements. Zero all other elements in the rows and columns occupied by that set. Continue this procedure by finding the next largest sets until all nonzero elements are accounted for as members of some set.
6. MAXMON
   Set to zero all elements except those

constituting a path or vector $E_{i_1 j_1} \ldots E_{i_d j_d}$ whose elements have a maximum sum under the constraint that $i_d < i_{d+1}$ and $j_d < j_{d+1}$.

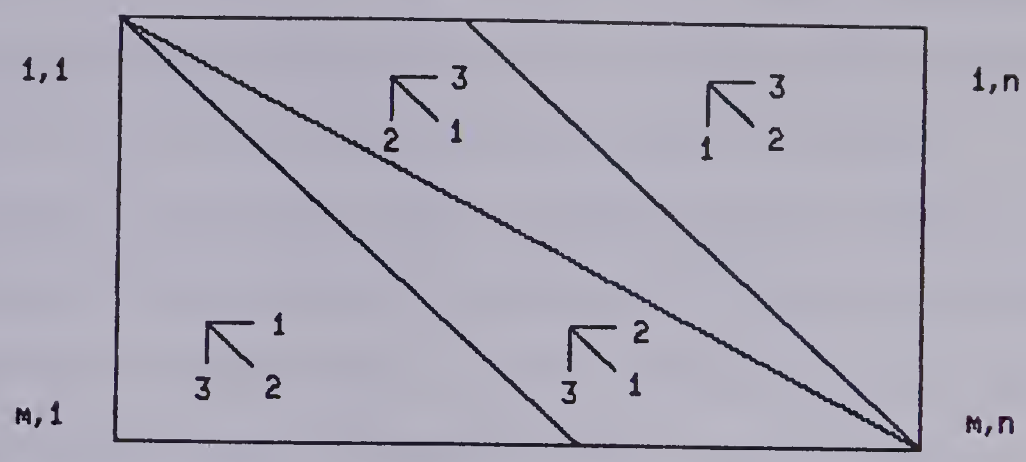
Similarity Phase

1. SUM
   Find the sum of the elements in the matrix.
   normalizing divisor: MAX(m,n).
2. DBL
   Find the sum of the elements in each diagonal. Multiply each sum by the length of the respective diagonal. Then obtain the sum of these products.
   normalizing divisor: $MAX(m,n)^2$.
3. PAIRS
   Calculate the sum of the products of all pairs of diagonally adjacent elements.
   normalizing divisor: MAX(m,n)-1.
4. STRING
   For every set of k diagonally adjacent elements, multiply the first member (that having the lowest indices) by k, the second member by k-1 and so on until the last member is multiplied by 1. Then sum all the elements in the matrix.
   normalizing divisor: $1/2(MAX(m,n)^2-MAX(m,n))$.
5. TRNSP
   For selected matrices only. Collapse the matrix by deleting all rows and columns containing only zeros. Let k be the number of row and column interchanges necessary to produce an identity matrix, or a matrix having all the nonzero elements in the axis. Calculate 1-k/(MAX(m,n)-1).

SORDER and SBYC is particularily detrimental in the case of SFIRST, which will be scuttled by an insertion appearing near the beginning of the vertical string or a deletion near the beginning of the horizontal string.

MAXMON resolves situations where more than one path is maximal by applying the selection scheme illustrated in

Figure 5: Resolving Directional Preference in MAXMON



Directional preference ranked 1, 2, 3.

Figure 5. In this scheme the preferred direction at any point in a path is determined by the current location in the matrix. In regions close to the axis, diagonal movement is most heavily weighted; and in outlying regions, movement toward the axis receives the greatest weight.

The LSTNG operation is appealing, not only because it does not require a weighting phase, but also because it can be simply described in English as the procedure of first finding the longest matching substring and continuing to find the longest matching substrings in the remaining positions.

The final phase of operation on the matrix returns a scalar which is r, the measure of similarity between the two strings. In these calculations, Alberga applied a normalizing factor to produce a value which ranged from 0.0

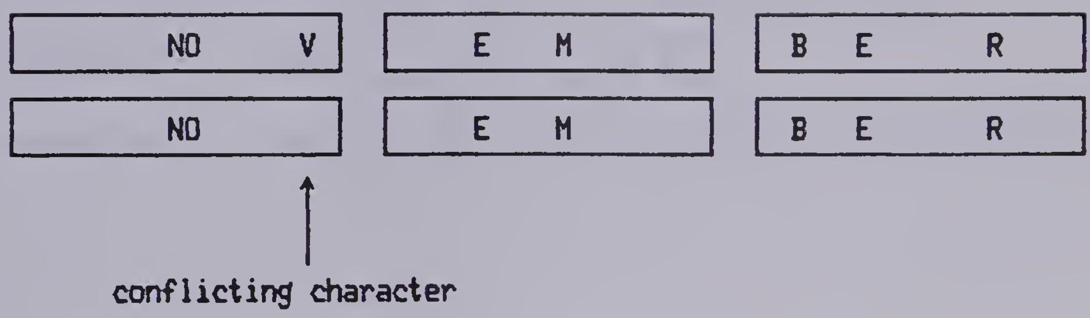
to 1.0. The normalizing divisors given in Table 9 apply only to matrices having no more than one nonzero element (selected matrices). Appropriate normalizing factors could not be found for unselected matrices. Instead, pre-normalized values were divided by the mean of pre-normalized similarity measures of each word with itself.

Szanser developed a string similarity technique which he has named *elastic matching* and has discussed in several documents (Szanser, 1969, 1971, 1973a, 1973b). Since it matches strings differing by one instance of character insertion, deletion, substitution, or adjacent transposition, it really amounts to being a rather speedy implementation of Damerau's algorithm.

The fundamental procedure in elastic matching is to first independently break both words into multiple "lines" as in Figure 6 where the word *november* is used. Each line has a fixed length equal to the size of the alphabet being used. Each position within a line corresponds to a member of that alphabet and can assume a boolean value. The particular character that a position represents depends on the ordering of the alphabet that has been chosen. In the examples which follow, the characters and order of the standard English alphabet are used. Strings are mapped on to a series of lines by giving the value 1 to the positions which represent characters present in the string and the value 0 to other positions. The order of the characters in the word is preserved by starting a new line whenever it conflicts with

Figure 6: Elastic Matching

original string ==> NOVEMBER
corrupted string ==> NOEMBER



the alphabetic order.

The elastic matching technique works well in the case where the number of lines obtained for both words is the same. A single exclusive or operation (XOR) gives the number of conflicting characters. Szanser has set a maximum threshold of one conflicting character permitted in a match but also noted that the method can be modified to support higher thresholds. If the algorithm is programmed at the machine level with each position in a line being represented by a bit, then one could expect very fast execution time. Furthermore, in CAI applications the author's target word could be mapped into the appropriate form at compile time.

Problems arise when an unequal number of lines is obtained as in Figure 7. If the number of lines formed from the two strings differ by more than one, then the procedure

Figure 7: Errors Causing Different Line Lengths in Elastic Matching

original string	==>	A J Q	M Z	J M Q	
deletion	==>	A J Q Z	J M Q		
insertion	==>	A J Q	M Z	J	J M Q

is abandoned, otherwise an XOR is applied to the paired lines. The first line in the student string which has conflicting bits causes the following lines to be moved ahead if the student string has fewer lines or moved back if it has more lines. A new matching process then commences at the first moved line with the previously matched characters masked out. If it results in no more than one conflicting bit then an approximate match is declared.

Wagner and Fischer (1974) have presented the most cogent general solution of the string similarity problem to date. Since it is based on an accounting of the edit operations necessary to convert one string into another, their analysis can be viewed as a generalization of the Damerau's work. They allowed the three edit operations of character deletion, insertion, and substitution, or, as

previously expressed, substitution operations on substrings of length 0 or 1.

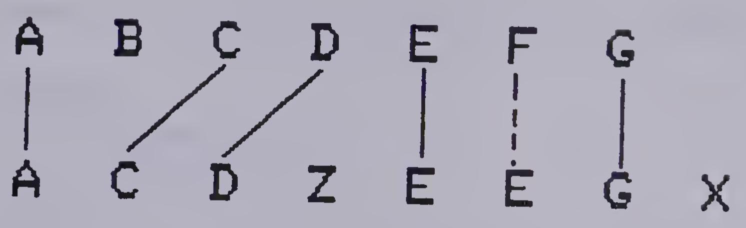
Each possible edit operation has a cost associated with it. The cost of an *edit sequence* is the sum of the costs associated with the series of operations comprising the sequence. They define "the *edit distance* from string A to string B [to] be the minimum cost of all sequences of edit operations which transform A into B".

The simplest weighting scheme assigns a cost of 1 to each operation except the substitution of a character with itself, which normally has a cost of zero. Wagner and Fischer note that a better approach is to weight operations inversely to their probability of occurrence in minimal cost edit sequences between words and their misspellings or mistypings:

> ... cost functions which depend on the particular characters affected by an edit operation might be useful in spelling correction, where for example because of the conventional keyboard arrangement it may far more likely that a character "A" be mistyped as an "S" than as a "Y".

Every edit sequence between two strings can be partially represented by a *trace*. In Figure 8, an example of the diagramatic expression of a trace, characters untouched by lines were, depending on the direction of the trace, either inserted or deleted. A dashed line indicates substitution of a different character and a solid line indicates substitution of the same character. One can think of a trace as "a description of how an edit sequence transforms A into B but ignoring the order in which things

Figure 8: A Trace Between Two Strings



happen and any redundency in" the edit sequence.

Wagner and Fischer proved that any edit sequence transforming string A to string B can be represented by a trace such that the cost of the trace is less than or equal to the cost of the edit sequence. Also, any trace from A to B represents at least one edit sequence having the same cost. Therefore, to find the edit distance, it is only necessary to find the trace of least cost. In the algorithm which accomplishes this, WAGNER (Table 10), idcost is a subordinate function which returns the cost of inserting or deleting the character passed to it. subcost is a function which returns the cost of a substitution operation involving the two characters passed as parameters.

Figure 9 Shows the matrix generated in the calculation of the edit distance between the word-misspelling pair of

Table 10: The WAGNER Algorithm In Pseudo-Pascal

```
A,B : string;
i,j,m,n: integer;
D[m,n]: array of integer;

***  first build the matrix D[m,n] ***

BEGIN
m := LENGTH(A);
n := LENGTH(B);
D[0,0] := 0;
FOR i:=1 TO m DO D[i,0] := D[i-1,0]+idcost(A[i]);
FOR j:=1 TO n DO D[0,j] := D[0,j-1]+idcost(B[j]);
FOR i := 1 T0 m DO
     FOR j := 1 TO n DO
     D[i,j] := MIN(D[i-1,j-1]+subcost(A[i],B[j]),
                   D[i-1,j]+idcost(A[i]),
                   D[i,j-1]+idcost(B[j]));

*** now output edit distance ***

PRINT D[m,n];
END.
```

circle vs. *curcal*. In the example, subcost is 0.7 and idcost is 0.5. The minimal cost edit sequence is indicated by lines connecting some of the matrix elements. A diagonal line indicates substitution, a vertical line indicates deletion of a character from the misspelling, and a horizontal line indicates insertion of a character into the misspelling. The resulting edit distance, found in the lower right matrix cell, is 1.7.

Lowrance and Wagner (1975) extended the calculation of edit distance to include transposition operations between any two characters. Hall and Dowling noted that the transposition of only adjacent characters is a special case of Lowrance and Wagner's extension and can be accounted for

Figure 9: An Example Showing the Calculation of Edit Distance

		C	I	R	C	L	E
	0.0	0.5	1.0	1.5	2.0	2.5	3.0
C	0.5	0.0	0.5	1.0	1.5	2.0	2.5
U	1.0	0.5	0.7	1.2	1.7	2.2	2.7
R	1.5	1.0	1.2	0.7	1.2	1.7	2.2
C	2.0	1.5	1.7	1.2	0.7	1.2	1.7
A	2.5	2.0	2.2	1.7	1.2	1.4	1.9
L	3.0	2.5	2.7	2.2	1.7	1.2	1.7

by simply adding to the minimization the expression: D[i-2,j-2] + COST(A[i-1],B[j]) + COST(A[i],B[j-1]). But since this would equate the cost of a transposed substitution to that of a simple substitution, it seems to the present author that either a constant should be added which represents the general cost of transposed substitution, or a table must be consulted to obtain the specific cost of the operation given the two characters in question.

The edit distance algorithm is appealing for several reasons. It is a well rationalized and intuitively reasonable formulation. Although clearly not as fast as some other methods, it has satisfactory computational bounds proportional to mn. Since it relies on a changeable data structure to assign weights to various edit operations, it

is flexible enough to support CAI applications in different natural languages, content areas, and so on. Perhaps its most important advantage in CAI is that, after the matrix is built, the trace of the edit distance can be obtained and used to show the student how to correct his response.

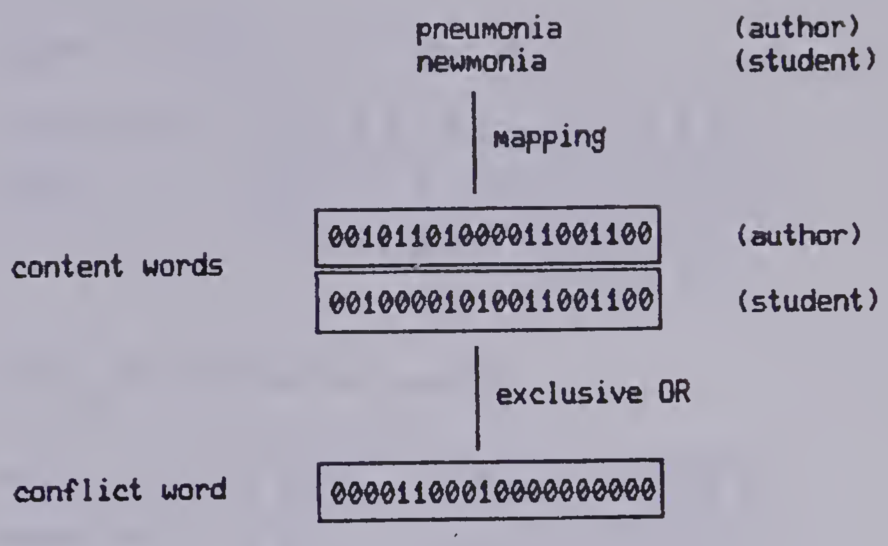
However, a major drawback to the edit distance algorithm as it currently stands is its inability to operate on parsed lexemes with lengths greater than one -- a fact which obstructs the recognition of many phonetically based errors. In addition, characters are not differentially weighted according to position.

Hybrid Approximate String Matching

A few reports exist of a hybrid approach to approximate string matching. In hybrid methods canonical forms of the original strings serve as operands to a similarity function. For example, Jackson (1967) described a system used in stock brokerage which reduced company names serving as index terms to a canonical form by vowel deletion and other means specific to the application, then performed a similarity-type matching operation between the index terms and abbreviated company names input by the user.

The most outstanding contribution to the application of approximate string matching to CAI has been that of the Computer-based Education Research Laboratory (CERL) at the University of Illinois -- the originators of PLATO. The principles outlined in a CERL report (Tenczar and Golden,

Figure 10: PLATO Word Recognition



1972) form the basis for the spelling algorithms used in PLATO at Illinois, CDC's commercial version of PLATO, and the TICCIT system now under the proprietorship of Hazeltine. Unlike most of the other methods summarized in this chapter, the PLATO word recognition algorithm was designed specifically to operate within the practical constraints of a time-shared CAI system *in vivo*.

In the PLATO method, both the author and student words are mapped to canonical forms called *content words*. Author words can be converted to content words before run time to enable sizable gains in execution speed. Content words have a fixed bit length which depends on the particular mapping algorithm used.

An XOR operation is performed on the two content words to obtain a *conflict word* which represents matched bits by 0

Figure 11: PLATO Letter Content field

```
machine ==>  | 1 0 1 0 1 1 0 0 1 0 0 1 0 0 0 1 |
masheen ==>  | 1 0 1 0 1 0 1 0 1 0 0 0 0 0 0 1 |
               E T A O N I S R H L D C U P F M
                           Q V G Y J K B W X Z
```

Figure 12: An alternate mapping

```
machine ==>  | 1 0 1 0 1 1 0 0 1 1 1 0 0 0 0 0 |
masheen ==>  | 1 0 1 0 1 0 1 0 1 1 0 0 0 0 0 0 |
               E T A O N I S R H D C U P F X G
                           Z L W M K Q B V Y J
```

and mismatched bits by 1 as in Figure 10. If the conflict word has all zero bits then the author and student words are judged to be an exact match. Otherwise, a binary search of a list of content words mapped from the 500 most frequent English words is performed to find one which equals the student content word. If such a match is found, the student word is judged to be different from the author word. When an exact match cannot be found with either the author word or any of the most frequent English words, then the word is judged to be a misspelling of the author word if the number of 1 bits in the conflict word does not exceed some fixed threshold.

The fundamental approach to the generation of content words proposed by Tenczar and Golden is the most significant and original feature of the PLATO spelling algorithm:

> Past attempts to devise methods for recognizing spelling errors have used a minimal set of human criteria (e.g., the phonetic approach). However, no one criteria appears sufficient to do the mysterious thing which humans do when they recognize words. Rather, we should use as many features of words as we can think of and hope that the interactions between these factors will contain the information that human beings use.

To implement this concept the content word is divided into fields, each representing some characteristic of the original string. The field lengths are determined in part by a "subjective feeling" for the importance of the characteristic which a field represents. The fields and their lengths suggested by Tenczar and Golden were:

1. Word Length (3)
2. First Character (4)
3. Letter Content (16)
4. Letter Order (10)
5. Syllabic Pronunciation (8)

They found that a 41 bit content word divided into the above fields produced satisfactory performance.

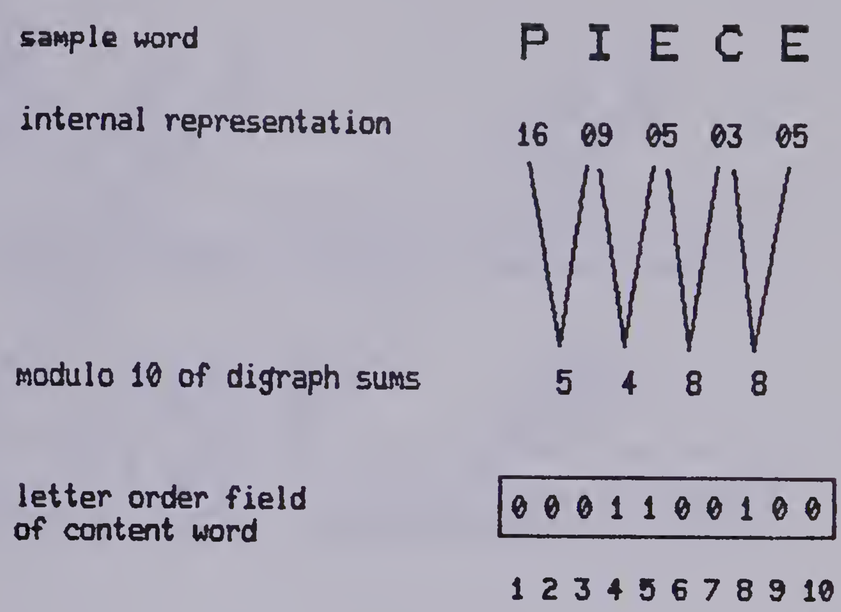
It was recognized that a standard binary representation of word length is inadequate because the number of conflict bits produced by an XOR operation would not be proportional to the difference in word lengths. With radix 2 representation, a word length of 1 (represented as 001) would produce only one conflict bit when compared with a word of length 5 (represented as 101). Instead, a coding scheme which attenuates or avoids this problem should be used. Tenczar and Golden put forth the *grey* or *unit-distance*

code (Bartee, 1972) as a likely candidate.

The introduction of a certain amount and type of ambiguity when mapping to the content word is an important feature of the PLATO spelling algorithm. For example, one bit in the First Character field is determined by whether the character is a consonant or a vowel. The specific character is then only approximately specified by the remaining bits.

The Letter Content field gives an approximation of which letters are present in a word without regard to their order and frequency. In this field insertions and deletions are likely to cause 1 conflict bit, substitutions result in 2 conflict bits, and transpositions will produce 0 conflicts. Figure 11 illustrates one scheme suggested by Tenczar and Golden for mapping the Letter Content field. In the method shown, savings in space were realized by mapping the 26 letter alphabet to a 16 bit field. This was accomplished by doubling up the less frequent characters on 10 of the bits so as to, one may presume, "heed the information theory dictum which states that in a coding scheme one should strive to have a probability of 0.5 of finding any given bit set". But the doubling of characters on bits can be put to a better use which the authors fail to note. The criteria for pairing characters could be determined by the frequency of substitution between characters in misspellings or frequency of their occurrence together as digraphs. Figure 12 gives an alternate mapping

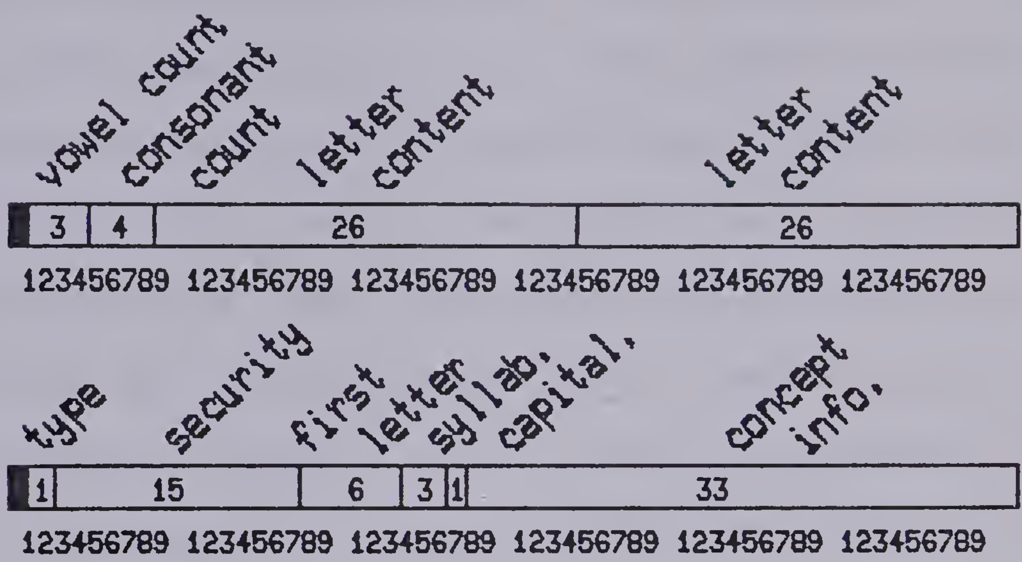
Figure 13: PLATO Letter Order Mapping



for letter content which may be more useful. In this alternate mapping scheme an attempt has been made to pair similar sounding letters together somewhat after the soundex method.

The letter order field is created by summing the internal values (perhaps ASCII or EBCDIC representations) of each pair of adjacent characters as in Figure 13. Where n is the length of the field, the values resulting from a modulo n operation on the sums are mapped into the field. Both adjacent character transposition and character substitution are likely to result in 2 conflict bits if they do not occur at the end or beginning of the word -- in which case they are likely to cause 1 conflict bit. Insertions and deletions are likely to cause 3 conflict bits if they occur internally but only 1 conflict bit when occurring at the beginning or

Figure 14: Current PLATO Word Mapping



ending of a word. More consistent results can be achieved by including terminators at the beginning and end of each word in the digraph mapping.

The assignment of the syllabic pronounciation field follows closely that of the letter order field. To approximate syllabification, consonant-vowel digraphs are identified, summed, and hashed into the field with a modulo operation.

A security field is used which contains enough additional information to uniquely identify any string. It is used in the initial comparisons to assure that two slightly different strings cannot map to the same content word and hence obscure the recognition of exact matches. This field is masked out of the conflict word before the conflict bits are counted.

Although the current implementation of the PLATO spelling algorithm[6] differs in detail from the method discussed by Tenczar and Golden, it is faithful to the general model they proposed. All CDC computers which run PLATO have a word size of 60 bits. Each content word occupies two 60 bit machine words divided into fields as shown in Figure 14. The first bits of both words are unused. In the first machine word, 3 bits are used to encode a count of the number of vowels, 4 bits contain a consonant count, and the remaining 52 bits are divided into two 26 bit letter content fields. To preserve some information about order, the first few letters are mapped into the first letter content field and the next few into the second. Any additional letters are coded in the former field. In the second machine word, 1 bit indicates whether the content word is representing a word or a number. In the case where a word is represented, 15 bits are assigned to the security field, the first letter is coded in 6 bits, 3 bits encode a count of the consonant-vowel pairs, and 1 bit indicates capitalization. Since the final 33 bits only hold information related to synonym comparison, the total number bits used for PLATO's spelling algorithm is 85.

Before a misspelling is declared, the following criteria must be met: the difference in vowel counts must not exceed one, the difference in consonant counts must not

----

[6] All information concerning the current implementation of PLATO has been obtained from personal communications with William Golden, head of the PLATO Services Organization at the University of Illinois.

exceed two, the number of conflict bits in the letter content fields must not exceed five, and the total number of conflict bits must not exceed some threshold.

The TICCIT version[7] of the PLATO spelling algorithm uses a three field content word:

1. First Character (5)
2. Digraphs (11)
3. Letter Content (16)

A comparison resulting in fewer than five conflict bits is judged to be a misspelling. Although the assignment of all three fields is similar to the equivalent fields discussed above, the value of the digraph field is obtained by ORing together entries from a 52 (26+26) word table indexed by character digraph sums.

Summary

CAI systems can and have made use of approximate string matching to identify incorrectly or alternately spelled words in student responses. Designers of CAI systems can benefit from existing literature documenting the research and application of approximate string matching algorithms.

String matching functions are of three types. Equivalence functions divide all possible strings into a finite number of mutually exclusive sets and return a binary value indicating whether two strings belong to the same set.

----

[7] All information regarding the TICCIT system was obtained from personal communications with David Stone, Instructional Design Supervisor with Hazeltine Corporation.

Similarity functions are generally more useful because they return a value indicating the proximity of two strings. Other approximate string matching functions are not symmetric and usually require some processing of the target word to be done by the course author. Hybrid functions are similarity functions where the similarity relation is taken between two canonical forms representing equivalence classes.

II. Experiments With Approximate String Matching

Originators of several of the approximate string matching functions described in Chapter I (Alberga, 1967; Blair, 1960; Damerau, 1964; Glantz, 1957; Symonds, 1970; Tenczar & Golden, 1972) attempted to measure the accuracy of the functions they proposed, and a few of them (Alberga, Damerau, Symonds) conducted comparative studies which examined differences between functions. The present chapter reviews the methods and results of these investigations and reports on research by this author aimed at:

1. replicating previously published findings
2. extending similar empirical assessment to the major functions reviewed in Chapter I
3. designing and assessing improvements upon the existing functions.

Earlier Studies

All previous tests of approximate string matching functions have made use of a list of paired strings. Such lists will be referred to here by the term *word data*. Word data lists can be formatted as two columns with the original strings, hereafter called *words*, in the left column, and the corruptions, misspellings, or alternate spellings, hereafter called *misspellings*, in the right column. The convention followed here will be that the number of distinct words in a word data list will be represented by M, and the number of pairs or misspellings by N. The four word data lists used by

the present author, three of which were taken from studies described below, appear in Appendix A. Long word lists of K strings, hereafter called *dictionaries*, were used by some researchers to measure type II error.

To test his abbreviation algorithm BLAIR, Blair used a word data list (M=N=117, see Appendix A) taken from a secretarial reference book (Hutchinson, 1956). The 4 character canonical form of each misspelling was compared with the 4 character canonical form of each word in the word data list. The case where a misspelling matched with several words was resolved by repeatedly incrementing the length of the canonical forms and taking additional passes through the list until either one or no match resulted.

In order to compare his single error matching function DAMERAU to BLAIR, Damerau used three word data lists: Blair's original word data, word data gleaned from newspaper articles during proofreading (M=41,N=44; see Appendix A)[8], and a much larger (M=N=964) word data list compiled from errors resulting from "equipment malfunction".

Damerau's method of comparison was fundamentally identical to that of Blair with the exception that a dictionary composed of K=1593 words randomly selected from text was merged with the correctly spelled words in the

----

[8] Two word-misspelling pairs contained in the original word data were deleted from the word data and results reported here: PIPE-LINE/PIPELINE because most of the functions discussed could be extended to handle hyphens but as proposed do not; and IZVESTIA/IZVESTIA, an identical pair unexplained by Damerau which may have occurred as a result of a case shift.

Table 11: Success and Error Frequencies Found by Damerau

Function		Word data Blair (N=117)	Damerau I (N=44)	Damerau II (N=964)
BLAIR	success	89 (76%)	32 (73%)	240 (25%)
	error I	26 (22%)	12 (27%)	676 (70%)
	error II	2 (2%)	0 (0%)	48 (5%)
DAMERAU	success	87 (74%)	36 (82%)	812 (84%)
	error I	30 (26%)	8 (18%)	122 (13%)
	error II	0 (0%)	0 (0%)	30 (3%)

larger (N=964) word data list to increase the type II error rates to measurable levels.

Blair and Damerau both expressed their results with three simple statistics: the number of correct matches, the number of failures to obtain any match for a misspelling, and the number of incorrect matches. The last two of these are related to type I and II errors respectively. Damerau's results, which replicated those of Blair, are summarized in Table 11.

The large difference between the type I error frequencies of the two algorithms with the Damerau II word data presumably arises from the fact that, while BLAIR was designed specifically to recognize corruptions introduced by humans, DAMERAU does not discriminate between the corruptions of different characters. Since the interest here is on corruptions from human sources, that is to say students, the results from the Damerau II word data can probably be ignored.

Alberga compared 65 distinct string matching functions. Of these: 58 were generated from his 13 operations described in Chapter I; 3 were Faulk's material, ordinal, and positional similarity functions; 1 was a phonetic matching function of Alberga's own invention; and the remaining two functions were those of Blair and Damerau.

As word data, Alberga used a sample from a body of misspellings collected by Masters (1927). Masters had grade 8, grade 12, and senior college students attempt the spelling of 268 words dictated to them. The resulting data consisted of a list of misspellings for each word with a frequency associated with each misspelling. In his thesis, Masters did not present the complete set of misspellings, but he did provide a list of those which were most frequent. This list was used by the present author and is given in Appendix A.

Alberga sampled Masters' original complete data by 1) expanding it to a list of word-misspelling pairs where each pair was duplicated according to the frequency associated with the misspelling and 2) selecting 1039 pairs from the expanded list. To test type II error, an additional 1039 pairs were generated by pairing each of the selected misspellings with a randomly chosen word from the original data.

The 65 matching functions were applied to both of the lists of paired strings. For any given function, a failure to return a match with a word-misspelling pair was counted

as a type I error. When a match was found with one of the random pairings, a type II error was tallyed.

Most of the functions tested by Alberga returned a value representing the similarity of the two strings. In these cases it was necessary to compare the similarity value against some threshold in order to determine if the strings matched. Therefore, a similarity function together with a threshold value can be considered to be an independent string matching function - which here will be represented as the name of the function followed by the threshold value in parentheses.

Table 12 shows some of Alberga's results. PHONE was a phonetically based equivalence function which, using rules requiring parsing, attempted to reduce each string to a canonical form representing its pronounciation. As simply the sum of the binary coincidence matrix, the function SUM is closely related to Faulk's material similarity function. Alberga's analysis of his results was based on the assumption that the two types of error are of equal importance. The functions were judged according to how well they minimized the expression

$$\sqrt{\mathrm{ERROR1}^2 + \mathrm{ERROR2}^2}$$

where ERROR1 and ERROR2 are percentage measurements of the two error types. Following this reasoning, Alberga concluded that the ROOF-SBYC-STRING(.12) function was the most successful and that, due to relatively large type I error rates, BLAIR, DAMERAU, and PHONE "failed rather badly".

Table 12: Error Frequencies Found by Alberga

Functions	N=1039 Error I		Error II	
BLAIR	346	(33%)	0	(0%)
DAMERAU	471	(45%)	0	(0%)
PHONE	375	(36%)	0	(0%)
SUM(.71)	44	(4%)	44	(4%)
ROOF-SBYC-STRING(.12)	22	(2%)	21	(2%)

Symonds tested several functions using two sets of word data. The first word data list, the results of a grade 5 spelling test, consisted of N=320 misspellings of M=100 words. The functions were tested by comparing every misspelling with every word.

Symonds compared her own function, BLAIR, and CONTEX-SBYC-PAIRS which she described as "one of Alberga's better methods". In fact, the latter function was never tested by Alberga. Symonds also failed to report the threshold at which this function was tested and the type II error found for this function and BLAIR. The results which she did provide are shown in Table 13.

The second word data list, whose source was not reported, contained 108 misspellings of 566 words, of which one may presume 458 were not paired with misspellings. Five functions were tested with this data: the 3 functions tested with the first word data list, DAMERAU, and DAMERAU-SYMONDS which worked by first attempting to get a match with DAMERAU and then applying SYMONDS if none was found. Although the method of testing was essentially the same as for the first word data list, the results were reported as in Table 14

Table 13: Error Frequencies Found by Symonds

Function	Error I (N = 320)	Error2
BLAIR	201 (63%)	-
CONTEX-SBYC-PAIRS	162 (51%)	-
SYMONDS	108 (34%)	4 (1%)

with type I and II error confounded. A quantity representing the accuracy of the function was incremented when a misspelling matched with the appropriate word but failed to match with any others.

Symonds concluded that the DAMERAU-SYMONDS function was "the best solution among those tested for the problem of automatic detection of misspellings".

Tenczar and Golden used a misspeller's dictionary as word data to measure the type I error associated with the PLATO matching function. Although they claimed that this source provided "common misspellings in English", dictionaries of this type usually provide only contrived phonetic misspellings of commonly misspelled words.

Type II error was measured by comparing successive pairs of words from a conventional dictionary. These methods produced measurements of 5% and 14% for type I and II error respectively.

Tenczar and Golden also used a rather novel testing strategy in which 50,000 strings of varying lengths were generated by a random process involving predetermined English letter frequencies. All $50{,}000^2$ possible comparisons between these strings were performed. Following the

Table 14: Success Frequencies Found by Symonds

Function	N = 108 Accuracy
BLAIR	83 (77%)
DAMERAU	85 (79%)
CONTEX-SBYC-PAIRS	90 (84%)
SYMONDS	97 (90%)
DAMERAU-SYMONDS	102 (94%)

rationale that pairs matched by the algorithm should look like misspellings of each other to human judges, these matching pairs were printed out and inspected. Although the number of these pairs was not given, Tenczar and Golden reported that only 50 pairs were judged to be not misspellings.

Difficulties which plague attempts to interpret the collective results of the above studies arise primarily from differences between the methods used in the individual studies. For example, the failure of the researchers, with the exception of Damerau, to use word data from other studies (and the failure to report word data used) naturally undermines comparison between studies.

A factor which further complicates interpretation for applications in CAI is the tendency of researchers to base their methods on a data retrieval model such as the following:

1. A data base is to be searched by the use of key words.
2. The key words in the data base are spelled correctly.
3. A user performing a search may enter misspelled keywords.

But in response analysis, the usual case is that a correctly spelled target word is compared with some number of misspelled or extraneous input words. Since most of the approximate matching functions under consideration are symmetric, this difference should not affect the measurement of type I error. However, the discrepancy between models does have implications for the measurement of type II error.

Finally, there seemed to be general confusion surrounding the importance of type II error relative to type I error. In most cases, type II error was either not reported, was confounded with with type I error, or was immeasureable due to an insufficiently large word data list. Alberga endangered the usefulness of his conclusions by using arbitrary comparison criteria which resulted in thresholds unsuitable for most applications. This was in spite of his recognition, as evidenced by the following statement, that an optimal balance of type I and type II error is highly application dependent.

> It should be noted that a solution satisfactory in one area may not be satisfactory in another. For example, in the airline systems, searches must be made for the records of passengers whose names have been misspelled on entry. In this case, one is willing to retrieve a number of wrong names as long as the right one is among them. Thus the threshold may be set fairly low. In computer-assisted instruction, on the other hand, one may want to be very certain of the match before one tells the student that he is probably right but seems to have misspelled his answer, rather than telling him he is wrong. In this application, therefore, the threshold may be set rather high.

This fact complicates the problem of comparing algorithms -- especially considering that even for specific applications

the optimal balance of errors is usually not known.

Method

A program of research was undertaken, the goal of which was to evaluate approximate string matching functions so as to determine those which would be most useful in a CAI environment. An attempt was made to model the method after the following application:

1. A number of target words have been entered into a CAI system by various authors.
2. Over time, the matching function is used on many occasions to compare each target word to several strings input by students.
3. Some of these strings are corruptions of the target word, but most are extraneous words.

Following this model, words in the word data correspond to target words, misspellings in the word data correspond to student corruptions of the targets, and the members of the dictionary correspond to extraneous words input by the students. Type I error was measured as the proportion of pairs from a word data list which the function failed to match. Type II error was measured as the proportion of comparisons between the words in the word data list and entries in a large dictionary which the function did match.

All programs were written in Pascal and compiled to native code to run on a Digital Equipment Corporation VAX 11/780 minicomputer under the VAX/VMS operating system. The

string matching functions were written as procedures in separately compiled modules and were called by various driver programs which managed all input, output and error tabulation. Due to the large number of string comparisons performed, the project consumed over 700 hours of cpu time.

It was recognized that the type of words and misspellings encountered by a string matching function in a CAI environment varies widely depending on the students, content, and instructional method. For example, one expects different kinds of corruptions or variations of the target word to arise from an anatomy course involving Latin (or latinized) terms in comparison with an English course for deaf students. Therefore it was decided to use word data from several different sources to partially guard against the possibility of general conclusions being drawn from unrepresentative word data.

The four word data lists used are identified in Table 15 and are presented in their entireties in Appendix A. The Blair, Damerau, and Masters word data lists are the same as those described earlier. The author collected the Nesbit word data by requesting Edmonton public school teachers (grades 2-6) to submit their students work in spelling tests.

The upper entry in each cell of Table 15 is either M or N depending on the column. For example, the Nesbit word data list contains 524 misspellings of 213 words. The second line in each cell contains the mean and, in parentheses, the

standard deviation of the string length.

The type of misspellings produced by an individual with a history of auditory experience of the original words, is likely to differ from the type produced had the history been dominated by visual experience. Similarly, misspellings occurring during dictation are likely to differ from those occurring when the word is copied from a visual model (for example, when a handwritten manuscript is typed). Misspellings in the Masters and Nesbit word data lists were the result of dictation. The status of the remaining two word data lists is not clear.

The *American Heritage Word Frequency Book* (Carroll, Davies, and Richman, 1971) was used as a source of extraneous words. It presents a list of 86,741 words drawn from 1,045 500-word samples of published text intended for children in grades 3-9. For this purpose a word was defined as any string of characters bounded by blanks. As a result, the list contains proper names, abbreviations, published misspellings and various oddities not found in conventional dictionaries. A standard frequency index (SFI) related to the estimated frequency of occurrence in the lexicon was calculated for each word.

The subset (K=21718) of this list used in the present study was determined by taking words with an SFI of 36.0 or greater, removing those containing non-alphabetic characters (except those containing apostrophies which in these cases were deleted from the word), converting all characters to

Table 15: Word Data Lists

	Words	Misspellings	Source
Blair	117 8.8 (2.0)	117 8.7 (1.9)	common misspellings
Damerau	41 8.2 (1.9)	44 7.8 (1.8)	newspaper errors
Masters	179 9.1 (2.3)	320 8.9 (2.4)	grades 8,12,16
Nesbit	213 6.1 (1.6)	524 6.2 (1.6)	grades 2-6

lower case, and finally removing any duplications resulting from the case shift.

During the measurement of type II error, dictionary words were tested against the target to determine if they were the identical, in which case the target-dictionary pair was not passed to the approximate matching function and an error was not counted. Unfortunately, there was no easy way of preventing words in the dictionary which might have been accepted by an author as misspellings, from being matched by the algorithm and counted as errors. To avoid this problem, it was decided to execute pilot runs with three algorithms and all four word data files. The algorithms (SOUNDEX, PLANIT, and DAMERAU) were chosen on the basis of *a priori* assumptions about their type II error rates (more liberal algorithms were desirable), their mutual distinctiveness, and speed of execution. The thousands of word pairs matched by the algorithms and counted as type II errors during these 12 pilot runs were printed out and inspected for pairs so

close in pronounciation or meaning that an author would accept the dictionary entry as equivalent to the target. To be more specific, the constituents of a pair were deemed equivalent if:

1. an orthographic interpretation was possible which produced pronounciations which were identical or differed by only minor nuance or inflection (e.g. size,sighs; does,dose,doze; massage,message)
2. their meaning was close due to a common root (accomodate,accomodation; field,afield) or they were synonymous.

As may be seen in Appendix A, all target words possess a / delimiter distinguishing an initial root from endings indicating grammatical function and so on. During the measurement of type II error, pairs were only passed to the approximate matching function if their roots did not match exactly. It was found that most of the problematic matches uncovered by inspection could be prevented by moving the delimiter to the left so that the pair shared the indicated root. The only way of resolving cases where the pair shared no initial root or where the common root was so short that many legitimate erroneous matches would have been blocked by shifting the delimiter, was to remove the dictionary word from the dictionary file. The deleted words are listed in Appendix B.

It was decided that the two initial goals, namely the replication of previous studies and the testing of untried

algorithms, could best be served by a single experiment comparing members of a selected set of algorithms by testing them with all four word data lists. The results of this experiment were then used to determine the nature of a second experiment aimed at assessing improved algorithms.

Experiment I

The nine algorithms compared in this experiment were selected from all those examined in Chapter I according to the following criteria:

1. feasibility of implementation. For example, the PLATO algorithm unfortunately could not be implemented because the information available was not complete enough to allow for an exact emulation.
2. inclusion in a previous comparative test. The large number of algorithms created by Alberga neccesitated the exclusion of all except that which he found to be most successful.
3. apparent potential. It was decided that there was no point testing algorithms which were clearly inferior (e.g. Glantz, Bourne and Ford).

The algorithms selected are listed in Table 16 in association with the mean execution time recorded for each word data list. Of course these times are not the shortest possible. They could presumably be minimized by coding in assembler or possibly by devising a more efficient high level implementation. The Pascal source code for each

Table 16: Mean Execution Times in Milliseconds

Functions	Word data Blair	Damerau	Masters	Nesbit
BLAIR	2.77	2.50	3.17	2.00
DAMERAU	0.14	0.14	0.11	0.14
SYMONDS	4.62	4.05	4.38	3.57
DAMERAU-SYMONDS	4.50	3.99	4.35	3.34
SOUNDEX	1.59	1.76	1.59	1.48
PLANIT	1.92	2.03	2.40	1.69
ROOF-SBYC-STRING	11.51	10.61	12.01	7.78
WAGNER	11.13	10.27	10.87	10.46
HALL	18.21	17.03	18.95	16.04

algorithm is shown in Appendix C.

Generally the versions of the algorithms used were the same as those proposed and tested by previous authors. Both BLAIR and SOUNDEX generated canonical forms truncated to four characters. The PLANIT canonical form was not truncated. The similarity metrics generated by ROOF-SBYC-STRING, WAGNER, and HALL were normalized to range as integer values from 0 (for dissimilar strings) to 100 (for identical strings). A vector of 101 elements representing the thresholds on the similarity value was updated after every comparison such that all elements having indices greater than (in the case of type II error) or less than (in the case of type I error) the similarity value were incremented. After all the comparisons had been performed, the vector contained the error frequencies associated with each threshold.

For both WAGNER and HALL, the cost of insertion or deletion (idcost) was set at 0.5 and the cost of substitution (subcost) was set at 0.7. Intuitively, a value

of subcost such that:

$$idcost < subcost < 2(idcost)$$

seemed appropriate. Although exceeding the upper limit would obviously result in no substitutions being performed, the effect of going below the idcost was not clear. Some pilot experimentation indicated that varying the subcost between these bounds had little or no effect.

HALL was an extension of WAGNER proposed by Hall and Dowling which allowed for the transposition of adjacent characters. Following the present author's suggestion in Chapter I, an additional transposition cost (tcost) was introduced so that the complete cost of such a transposition would be:

$$subcost(i,j-1) + subcost(i-1,j) + tcost$$

where tcost was arbitrarily set at 0.2.

Figures 15, 16, 17, and 18 show the results for each word data list. Type I error (on the vertical axis) is represented as a percentage calculated by:

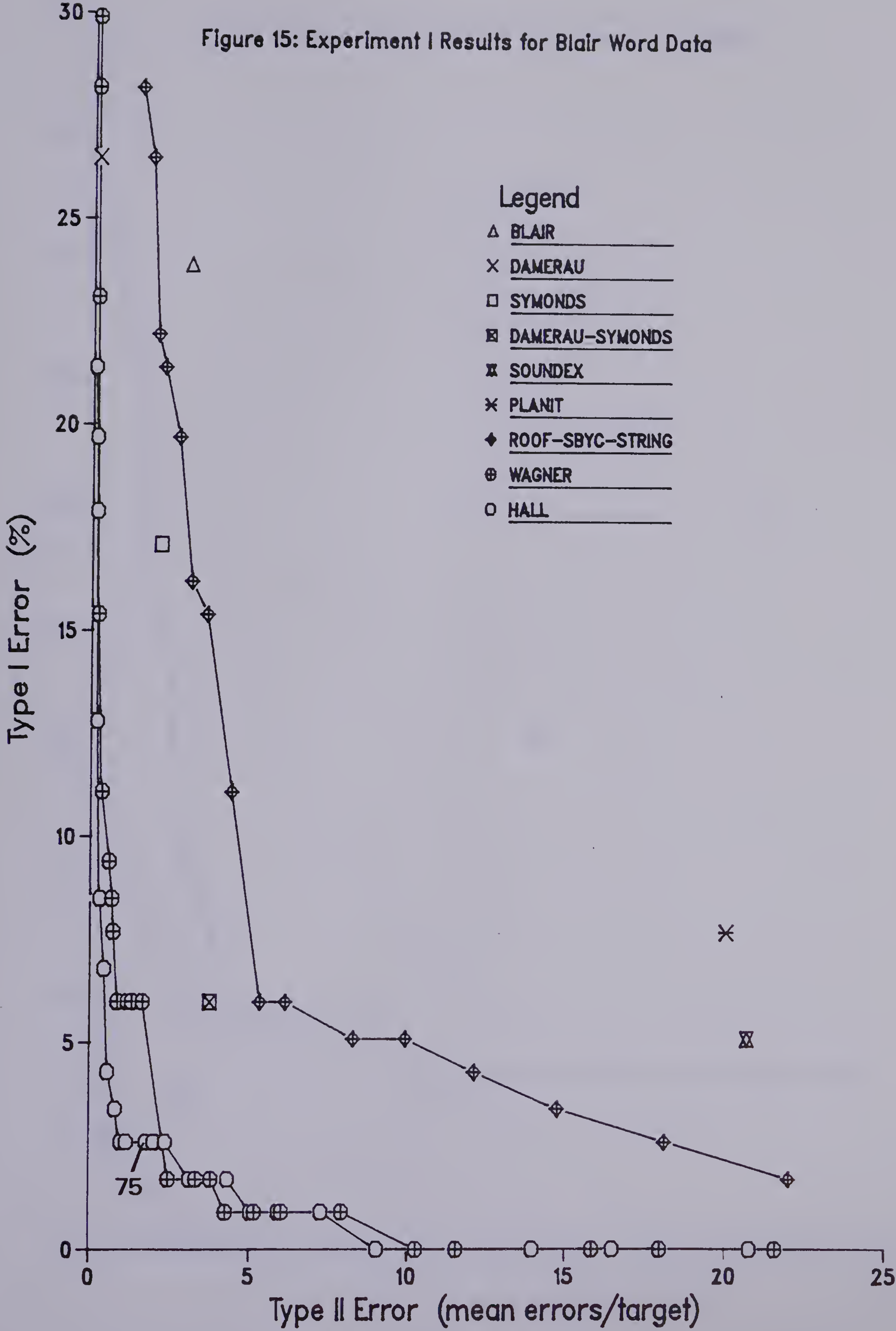
$$100(error1/comparisons)$$

where comparisons = N. Type II error (on the horizontal axis) is represented as the mean frequency of errors per target word calculated by:

$$error2/(comparisons/K)$$

where comparisons = MK. Note therefore that readings on the horizontal axis differ from percent of type II error by a constant factor of 100/K.

Figure 15: Experiment I Results for Blair Word Data



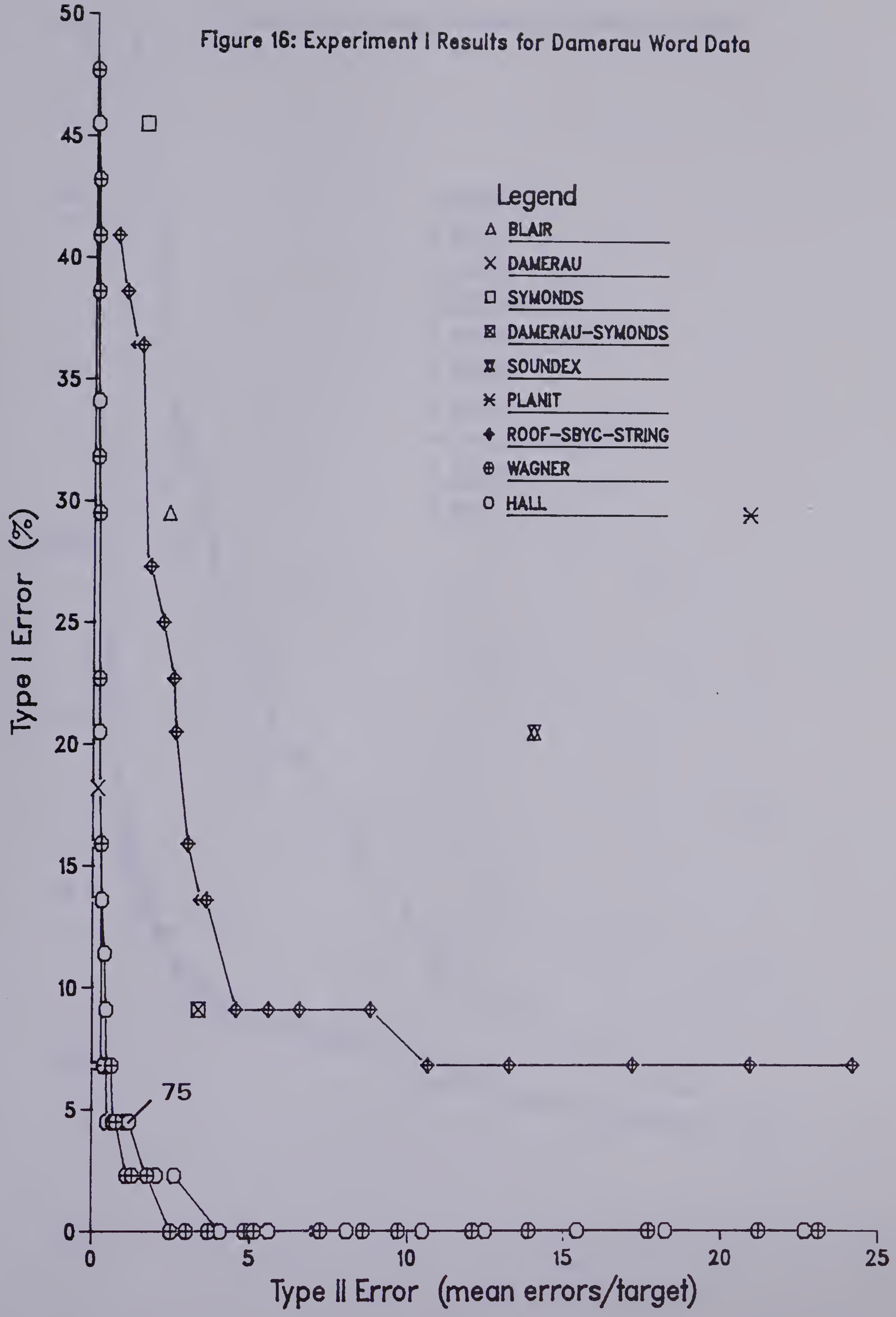


Figure 16: Experiment I Results for Damerau Word Data
Legend
BLAIR
DAMERAU
SYMONDS
DAMERAU-SYMONDS
SOUNDEX
PLANIT
ROOF-SBYC-STRING
WAGNER
HALL
75
Type I Error (%)
Type II Error (mean errors/target)
0
5
10
15
20
25
30
35
40
45
50

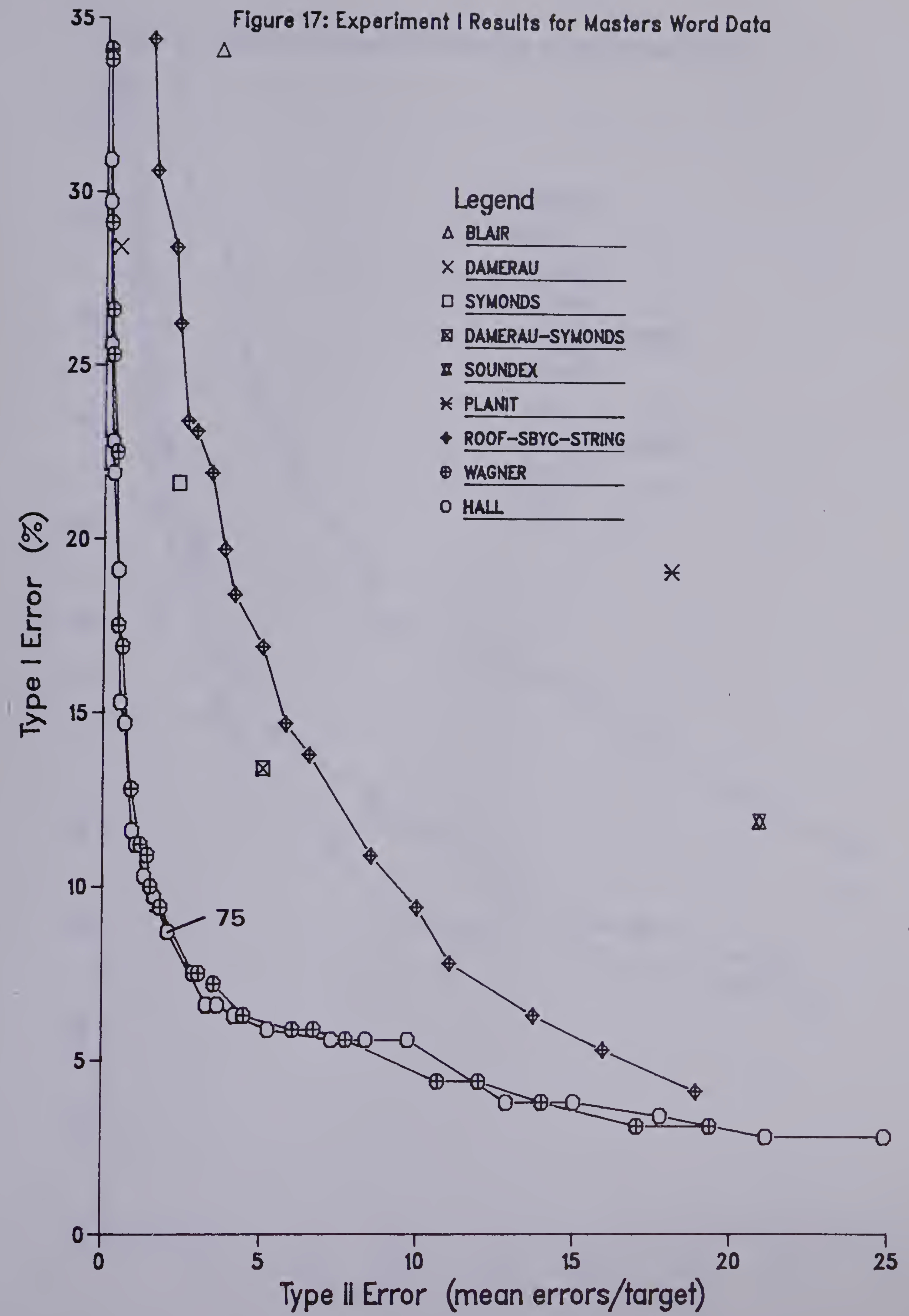


Figure 17: Experiment I Results for Masters Word Data
Legend
BLAIR
DAMERAU
SYMONDS
DAMERAU-SYMONDS
SOUNDEX
PLANIT
ROOF-SBYC-STRING
WAGNER
HALL
75
Type I Error (%)
Type II Error (mean errors/target)
0
5
10
15
20
25
30
35

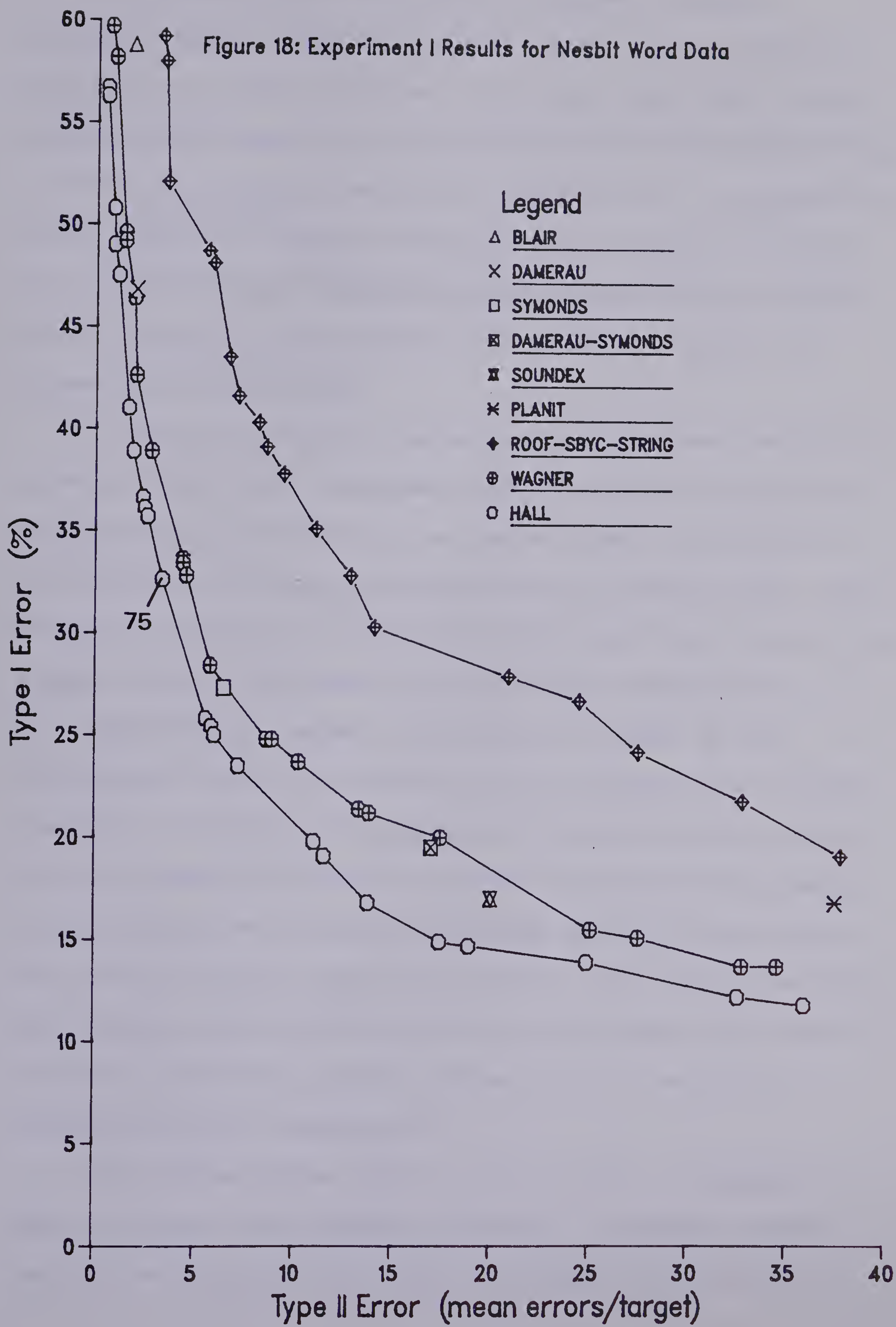


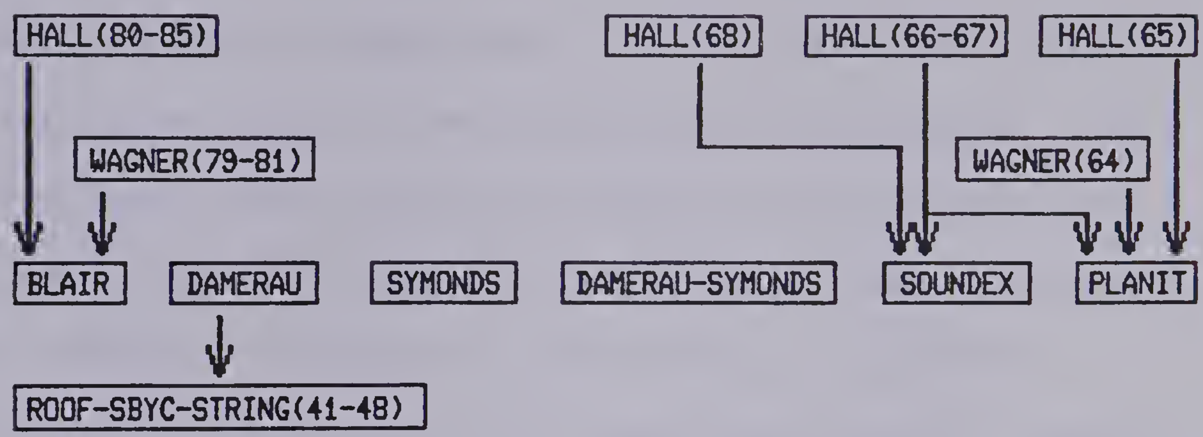
Figure 18: Experiment I Results for Nesbit Word Data
Legend
BLAIR
DAMERAU
SYMONDS
DAMERAU-SYMONDS
SOUNDEX
PLANIT
ROOF-SBYC-STRING
WAGNER
HALL
75
Type I Error (%)
Type II Error (mean errors/target)
0
5
10
15
20
25
30
35
40
45
50
55
60

The binary functions (BLAIR, DAMERAU, SYMONDS, DAMERAU-SYMONDS, SOUNDEX, PLANIT) appear on the graphs as single points. The non-binary functions (ROOF-SBYC-STRING, WAGNER, HALL) appear as connected points, each representing a distinct threshold setting of the function. In Appendix D, which shows the complete results of experiments I and III, can be found the threshold settings associated with each point, as well as the results for extreme settings not appearing on the graphs.

A system of ranking the algorithms had been settled on *a priori* which was independent of the relative importance which may be placed on the two error types. According to this system, an algorithm could only be ranked higher than a second algorithm if it bettered that algorithm in both error types over all word data lists. This was applied to ROOF-SBYC-STRING, WAGNER, and HALL as if each of the thresholds tested with these functions identified a single function. In Figure 19, a diagram illustrating the results of this ranking, connecting arrows indicate a better⇒worse relationship. Unconnected algorithms share the same rank. This diagram shows comparisons between the binary functions and between them and the non-binary functions, but does not cover the numerous possible comparisons of non-binary algorithms among themselves.

The diagram shows that no one instance of WAGNER or HALL is better than DAMERAU, SYMONDS, or DAMERAU-SYMONDS over all word data lists. This is in spite of the fact,

Figure 19: Experiment I Comparisons Over All Word Data



illustrated by the 4 previous figures, that there are instances of HALL which, on all word data lists taken singly, are better than both SYMONDS and SYMONDS-DAMERAU. The anomaly is caused by a shifting of WAGNER and HALL data points between word data lists, particularly between the Nesbit word data and the others.

Referring back to the previous figures, note that one of the instances of HALL has been marked with its threshold value -- 75. Observe further that this point remains within the rectangles defined by SYMONDS, DAMERAU-SYMONDS, and the origin in all cases except with the Nesbit word data list. Although it can not be determined with certainty which characteristics of the Nesbit word data are responsible for this interaction, perhaps the simplest hypothesis is that shorter string length produced a greater increase in type I

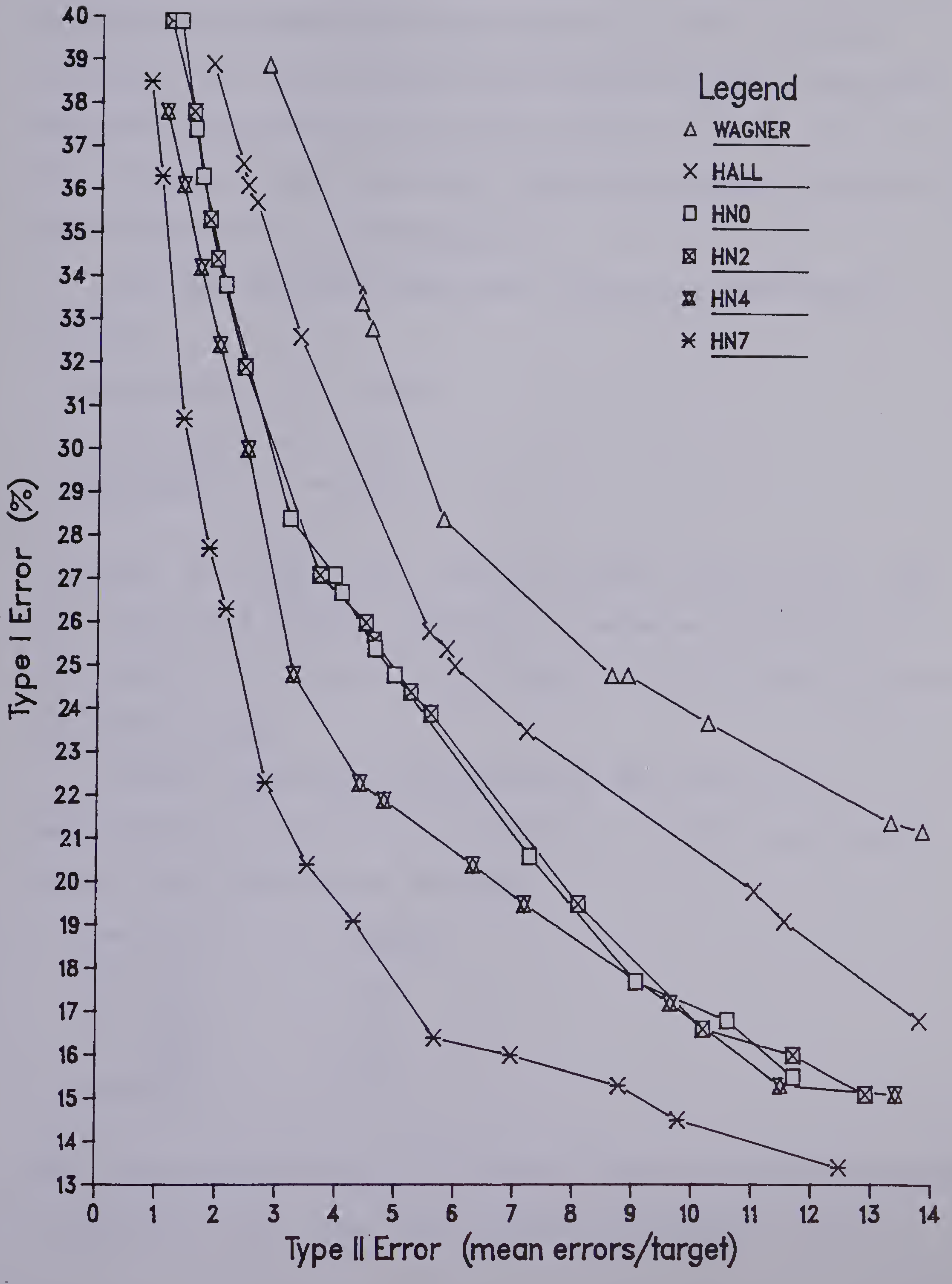
error in HALL than in the binary algorithms.

Experiment II

The results of experiment I indicated that HALL had the greatest potential as a starting point from which a very accurate approximate matching function might be produced. This was especially apparent considering that, while it is easily modified by altering the costs of its edit operations, it had as yet not been specifically tuned to account for orthographically based misspellings which are so common. It was decided that HALL would be modified and tested in a stepwise fashion using the Nesbit word data. The modified versions, identified as HN0 through HN7, are plotted in Figure 20. Again, each point represents the type I and type II error resulting from a single threshold setting.

HN0 was identical to HALL with the following exceptions. A value of 0.5 was added to idcost if the character in question was in the initial position. A value of 0.1 was added to subcost if one of the characters involved was in the initial position and 0.2 was added if both were in that position. These values were arrived at by inspecting a small sample of dictionary words that HALL erroneously matched by editing the initial character, and calculating the additional weights required to lower their similarity values below the threshold. When tested, HN0 was found to better the performance of HALL over all thresholds.

Figure 20: Experiment II Results



HN1 was identical to HN0, except that all identical adjacent characters in both strings were deleted before the edit distance matrix was constructed. Since this modification produced no improvement over HN0, another version, HN2, was created which instead deleted identical adjacent occurrences of only the letters l, m, n, p, r, t. The failure of HN2 to produce improved accuracy resulted in the abandonment of this approach.

HN3 was similar to HN0, but idcost was modified as follows:

character	idcost
e	0.3
a,i,o,u,h	0.4
otherwise	0.5

Although the values were selected rather arbitrarily, the rank order was based on statistics gathered by Masters on the frequency of insertion and deletion of different letters in misspellings.

Modest improvements produced by HN3 led to the development of HN4 which was identical to HN3 except that subcost was modified as follows:

character	subcost
b,f,p,v	0.6
d,t	0.6
c,k,q	0.5
c,s	0.4
otherwise	0.7

With the exception that, of course, the cost of substituting a character with itself was always 0.0 the above costs

should be interpreted such that the subcosts in the right column apply to substitution between characters within the group indicated on the left. HN4 produced the best results of all previous attempts.

HN7 was identical to HN4 except that a lower cost (0.4) was introduced for substitutions between vowels. This resulted in marked gains over the performance of HN4. Appendix E lists actual type I and type II errors committed by HN7 with the Nesbit word data.

Experiment III

HN7 was tested with the remaining word data lists to determine if it, being more accurate than HALL, would have any single threshold settings which bettered DAMERAU, SYMONDS, or DAMERAU-SYMONDS over all word data lists. The threshold settings found to satisfy this condition were 78-80, which bettered SYMONDS, and 75-77 which bettered DAMERAU-SYMONDS. No instances of HN7 were found to better DAMERAU over all word data lists.

III. Prospects for Approximate String Matching

This concluding chapter briefly examines how approximate string matching functions can be chosen for, and put to work in, CAI systems. Factors to be considered in the selection of these functions are identified. There is a short discussion of response markup and dictionary support services. These facilities make use of approximate matching but go beyond the fundamental response analysis problem which was the concern of the previous chapters. Finally, areas which would benefit from the attention of further research are identified.

Selection Factors

Of course, there exists no completely determined and objective rationale for deciding which functions are most appropriate and how they are best implemented. Although a publicly accepted set of factors upon which to base these decisions can probably be defined, individuals will disagree as to the weight carried by each factor in the selection process. What follows is an admittedly non-orthogonal list of six selection factors accompanied by comments on the implications and relative importance of each factor.

Accuracy

As the subject of the last chapter, accuracy is the selection factor about which we have the most information. Conclusions based on the superior performance of the edit distance algorithm are weakened

by the fact that several of the the functions reviewed in Chapter I, including the PLATO algorithm, were not tested. Furthermore, it must be recognized that the findings of Chapter II are dependent on the word data used. There may indeed exist some application for which BLAIR is more accurate than HALL.

Speed

Not unexpectedly, the execution speed of an algorithm was found to be inversely related to its accuracy. DAMERAU turned out to be very fast, a fact supporting its selection for use in operating systems, compilers, and the like. Although the most accurate algorithm, HN7, was also the slowest, there are a number of techniques which could be applied to shorten the execution time of all versions of the edit distance algorithm.

1. Threshold Pruning.

   In CAI, we are not usually interested in the edit distance *per se*, but only whether it exceeds a given threshold. In this case the threshold can be used, while the matrix is being built, to avoid processing elements which are predicted to exceed the threshold. The amount of time saved by threshold pruning will, therefore, be dependent on the threshold value used.

   When the algorithm can determine that D[m,n] must exceed the threshold, then it is able to

immediately return a no-match result to its caller. Adhering to this principle, the simplest threshold pruning modification is to check each completed column (where the matrix is built column by column) to ensure that it contains at least one element not exceeding the threshold. If so, it moves on to the next column. But if not, matrix processing ceases and the match fails. Preliminary experimentation with the word data used in Chapter II, indicates that this modification can yield a time saving of about 50% for comparisons resulting in failure to match.

To achieve time savings for successful comparisons, it is necessary to employ a technique which prunes diagonally. It is worth noting that such techniques will yield lesser savings when applied to algorithms allowing adjacent transposition than when applied to those which allow only insertion, deletion, and substitution.

2. Length Difference Test.

Since very few misspellings differ in length from the original by more than 2 characters, a test of length difference can be applied before entering the edit distance algorithm to exclude pairs which are unlikely to match. Of course, savings will only be realized for those pairs which are excluded by the test. Preliminary investigations excluding pairs

whose length differed by more than two characters, yielded mean savings of about 40% over all comparisons which resulted in no match. No loss of accuracy was detected.

3. Content Difference Test.

   Following the same principle as the length difference test, content fields, similar to those used in the PLATO algorithm, could be exclusively ORed together to find the number of characters found in only one of the two strings. Pairs differing by more than some fixed number of characters could be rejected without further processing. Although this test is likely to be more successful at weeding out dissimilar pairs than the length difference test, whether its superior discriminatory powers justify the time expended in the generation of content fields is an empirical question.

4. Truncation.

   By truncating both strings to some maximum length, an upper bound can be imposed on the execution time. In practice, since execution time increases as the product of the string lengths, truncation will probably be necessary to prevent abuse. Preliminary investigations which truncated both strings to six characters, found savings of about 30% unfortunately accompanied by a considerable loss of accuracy. It now seems likely that truncation to lengths of 8-10

characters would be more appropriate.

5. Combining Time Saving Techniques

It is probable that a combination of the above techniques would result in the best performance. Also, the speed of DAMERAU makes it a candidate for use as a kind of pretest. The following example in pseudo-Pascal shows what a time optimized design might look like. EDITDIST is a version of the edit distance algorithm which truncates both strings to 10 characters and uses threshold pruning while building the matrix. Both EDITDIST and DAMERAU are functions which return a result of either MATCH or NOMATCH. RETURN transfers control back to the calling program with a returned value of either MATCH or NOMATCH.

```
BEGIN
diff := |LENGTH(stringa)-LENGTH(stringb)|;
CASE diff OF
>2: RETURN(NOMATCH);
 2: RETURN(EDITDIST(stringa,stringb));
<2: IF DAMERAU(stringa,stringb)=MATCH
        THEN RETURN(MATCH)
        ELSE RETURN(EDITDIST(stringa,stringb));
END;
```

Program Size

Of the functions reviewed in Chapter I, The PLATO algorithm with its 500 word dictionary was probably the largest. Of those tested, SYMONDS and DAMERAU-SYMONDS were the largest. However, none of these can be considered prohibitively large, and size can probably be

disregarded as a discriminatory factor.

Ease of Implementation and Maintenance

Programmers are expensive. Algorithms which require a machine level implementation, such as the PLATO algorithm, are likely to be harder and more costly to install, maintain, and port to other systems. The readability of the implemented program, whether it be in assembler or a high level language, is also important. Bugs are not readily apparent in programs whose source code is, perhaps of necessity, complicated and unclear. This factor has special importance in approximate string matching, since bugs which hamper the accuracy of the algorithm may go unnoticed by users.

Adaptability

As has been observed earlier, different CAI applications have different response analysis requirements. Approximate string matching algorithms which can be adapted to specific applications by minor modification or by the passing of parameters, have a clear advantage over those which cannot. The edit distance algorithm is by far the best in this regard. Adjustments in the relative probabilities of type I and type II error can be made by varying the threshold value passed as a parameter. Adaptions to diverse natural languages, can be achieved by modification of the edit costs.

Response Markup

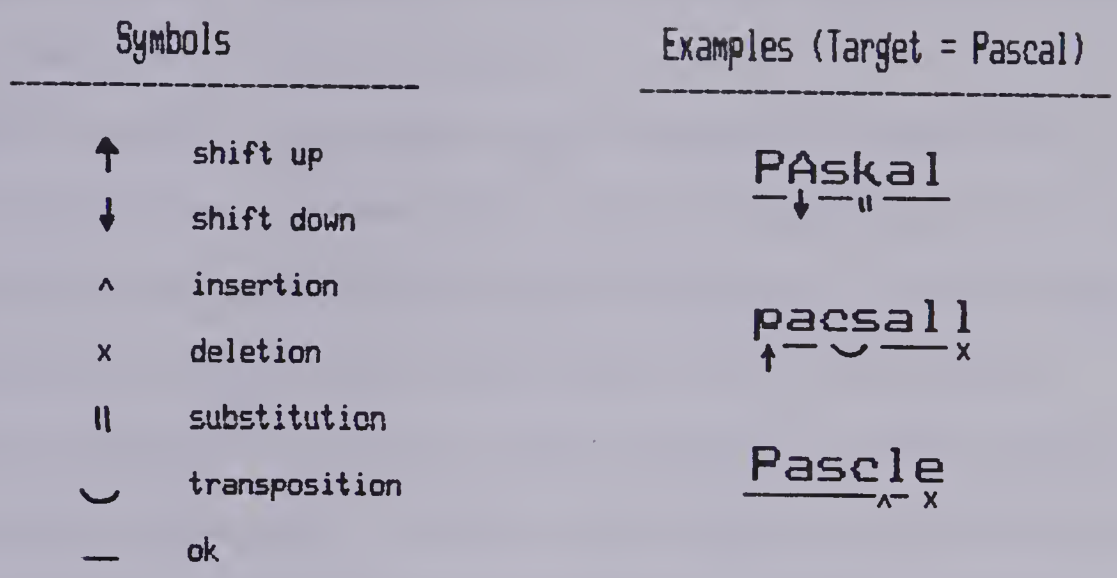
In PLATO, response markup is a facility which informs the student about discrepencies between his response and the author's target by writing special symbols on the screen below the entered response. PLATO response markup indicates unanticipated words, words not in correct order, and the locations at which words are missing. The only information available about the internal form of a word is whether it is misspelled and whether the initial letter should be capitalized.

The edit distance algorithm can provide more complete support for response markup. By tracing the edit sequence back through the matrix, enough information can be obtained to show a student exactly how his misspelling can be corrected. If small substitution costs are introduced for case shifts, instances of inappropriate letter case can be indicated as well.

Figure 21 shows some proposed symbols for response markup and gives some examples of how they would be used. Correctly spelled words matching the target would be underlined to distinguish them from unanticipated words, which would be left untouched. Only anticipated correct responses would be subject to markup. It would seem pedagogically unsound to provide markup for anticipated incorrect responses.

An alternative method of response markup is to process each word in the response after it has been entered but

Figure 21: Response Markup



before the student signals the completion of the response by pressing the enter key. This way, the student's spelling errors are brought to his attention, and perhaps corrected by him, immmediately after they are committed.

Depending on the author's goals, one of the following strategies might be used after a misspelled word has been detected and marked:

1. The student is free to either go back and correct the word, or to ignore it and continue entering his response. When the enter key is pressed, all outstanding misspellings are automatically corrected on the screen.
2. The student is forced to go back and correct the misspelling before continuing. The cursor is moved sequentially to each point in the misspelling at which a correction is to be made. Whereas the strategy in point

one may incline the student to continue misspelling the same word in later responses, this method, much like a human tutor, imposes a cost on spelling errors which the student will try to avoid.

3. The student must press some key which causes the word to be corrected automatically and allows him to continue. A compromise between the above methods, this strategy imposes a small cost (one key press) on errors.

According to a recent advertisement in the popular microcomputing press[9], Tenczar and others have implemented an extension of BASIC for the Apple II which provides support for textual response analysis including a response markup facility virtually identical to the one proposed above. This implies that they have used the edit distance algorithm, or something very similar.

Dictionary Support

Several features can be identified which would be desirable additions to CAI systems, but which involve searching, sometimes for an approximate match, through relatively lengthy lists of words referred to here as dictionaries. This section consists of a brief description of these features followed by a tentative and cursory design expressing them as operations on a common set of dictionary structures.

----

[9] EnBASIC by the Computer Teaching Corporation advertised in *MICRO*, April 1983.

The grammar and spelling correction capabilities of modern word processors would be extremely useful during the creation of instructional frames of text, one of the most time consuming tasks of course authoring. The spelling correction facilities of word processors search for every word parsed from the document in a large (10,000-100,000 word) dictionary representing the English lexicon. If a word cannot be found, true spelling correctors present to the author the closest approximation found in the dictionary, which is presumed to be the correct spelling of the unidentifiable word. The author then has the choice of:

1. replacing the word by its approximation
2. entering the word into the dictionary
3. ignoring the discrepancy and continuing
4. deleting the word and trying to correct it himself.

Another desirable feature requiring dictionary searching is an online source of standard dictionary definitions, etymologies, synonyms, antonyms, and so on. Both students and authors should be able to use this sort of facility. When the word supplied by the user cannot be found in the dictionary, the closest approximations found should be presented as alternatives from which a choice can be made.

Returning to the realm of response analysis, a synonym matching facility, similar to that supported by the -vocabs- and -concept- commands in TUTOR, will be an important part

of future CAI systems[10]. In using such a feature, the author specifies, perhaps by default, synonyms associated with target words. Student responses exactly or approximately matching the synonym would be treated as if they matched the original target.

Finally, recall that in the PLATO approximate string matching algorithm, a response word not matching exactly with a target word was first searched for in a dictionary containing the 500 most common English words. If it matched exactly with any words in this dictionary, it was rejected as a potential approximate match. This scheme is probably an effective way of reducing type II error and can easily be applied to any approximate matching algorithm.

How can we design a system which integrates all of these features as operations upon a common set of dictionary structures? In *Computer Programs for Spelling Correction*, which is concerned with the design of an interactive word processor type spelling corrector, Peterson (1980) provides us with at least half the solution. He noted that, according to the Brown corpus analyzed by Kucera and Francis (1967), the 256 most frequent words in the English lexicon account for over 55% of specific instances of usage[11]. Peterson also observed that for any specific document, a relatively small group of words exist which occur frequently in the document

----

[10] Among others, the WISE authoring system from WICAT and EnBASIC both support synonym matching.
[11] This is a manifestation of a phenomenon known to statistical linguists as Zipf's Law (Zipf, 1949).

but infrequently, or perhaps not at all, elsewhere. This led him to propose the following searching strategy:

> First, search the small table of most common English words.
>
> Next, search the table of words which have already been used in this document.
>
> Finally, search the large list of remaining words in the main dictionary. If a word is found at this level, add it to the table of words for this document.

Adapting Peterson's proposal to the problem at hand, one might consider using the following four data structures:

- Common word list (about 256 words)
  - contains the most frequent English words
  - structured for fast exact match searching
  - resident in main memory, or in a localized area of virtual memory

- Course word list (less than 2000 words)
  - one for each course
  - contains all words entered to text or glossary
  - structured for exact and approximate match searching
  - resident in main or virtual memory

- Master Dictionary (20,000 - 100,000 words)
  - contains many English words (including those in the common words list) with associated definitions, synonyms, etc.
  - structured for exact and approximate match searching
  - controlled by the system manager
  - resident on disk

- Glossary (less than 2000 words)
  - one for each course
  - contains words chosen and defined by the course author, with synonyms, etc.
  - structure similar to master dictionary
  - resident on disk

As in Peterson's strategy for spelling correction, text words entered by the author are searched for first in the common words list, then in the course words list, and

finally in the master dictionary. If this exact match search fails, a search is performed on the course words list and the master dictionary to find the closest approximation. If the author decides to keep the unidentified word, it is inserted into the course words list. He may decide that it should also go into the glossary, in which case he must supply a definition, synonyms, and so on.

When an author is specifying a target for which synonym matching is to be allowed, synonyms drawn from the master dictionary and glossary are displayed. Appropriate synonyms are then selected by the author and copied into some separate structure used for response analysis.

The master dictionary and glossary would be used by authors and students in much the same way as one would use their paper counterparts. Searching with an approximate key would be possible. The author would be able to alter the contents of the glossary at any time.

In response analysis, when a response word failed to match the target exactly, the common word list would be searched for an exact match before an approximate match would be attempted. As described earlier, if an exact match was found an approximate match with the target would not be permitted.

The success of designs of the type described in this section rest on the efficiency of the search algorithms used. Algorithms which efficiently search for only an exact match are commonplace, and may be found in texts such as

Knuth (1973). There is no reason to believe that the available algorithms are not up to the task. However, searches for approximate matches in structures as large as the proposed master dictionary, are certain to be too slow for use in truly interactive systems if those searches are performed sequentially. This assessment is based on the assumption that the maximum time period users can be expected to happily wait for a search is somewhat less than the time it would take them to do it manually with a paperback dictionary.

Fortunately, a few investigators have proposed methods which avoid a sequential search for an approximate match by imposing some special structure on the dictionary. Mor and Fraenkel (1982) described a hash table implementation of Damerau's method which appears to be very fast. Kashyap and Oomen (1981) described an efficient version of the edit distance algorithm which represents the dictionary as a tree structure such that any information obtained during the evaluation of any one edit distance, which may be relevant to the computation of other edit distances, is not wasted.

Further Research

Increases in the accuracy of the edit distance algorithm can, no doubt, be realized by adjusting edit costs to more appropriate values than those used in HN7. Some systematic procedure for achieving this would be preferable to the rather arbitrary method practised by the present

author. One interesting possibility would be to start with the edit costs used in HN7 and introduce an adaptive mechanism such that, while calculating the edit distance between a large number of word-misspelling pairs, the costs of operations appearing in minimal cost edit sequences are decremented relative to the costs of other edit operations. If this were counterbalanced by an incrementing of costs of operations occurring in minimal cost edit sequences between randomly selected word-word pairs, the costs might converge to optimal values.

Another area of potential improvement is the normalization factor which converts the absolute edit distance into a value expressed relative to string length. It may be that by modifying this factor to improve the robustness of the algorithm to variations in string length, the shifting of data points reported in Chapter II would be reduced.

Several other modifications to the edit distance algorithm can be imagined which might increase its accuracy. One could first parse each string into phonetically meaningful lexemes (e.g. *th*, *ph*, *ght*, *c*, *x*), and use these instead of characters as the subjects of the edit operations. Another possibility is to incorporate more positional information such that, for example, *e* would have a lower deletion cost were it in the final position. Unfortunately, these sorts of modifications could not be made without considerable increases in execution time.

Improvements which would have the greatest practical importance would be those which reduced execution time without sacrificing accuracy. Experimentation with the time saving techniques described earlier would contribute to this objective.

As CAI becomes widespread, realistic word data upon which approximate string matching functions can be tested will become readily available. In addition to supporting the further improvement and tuning of these functions, this will clarify our understanding of the degree to which specialized functions are necessary for different content areas and student types.

The next major advancement in the solution of the approximate string matching problem will probably be the introduction of contextual information. This will mark the beginning of an inevitable merger of this problem with the greater problem of natural language understanding.

Bibliography

Alberga, C. N. String similarity and misspellings. *Communications of the ACM*, 1967, 10(5), 302-313.

Bartee, T. C. *Digital computer fundamentals*. New York: McGraw-Hill, 1972.

Blair, C. R. A program for correcting spelling errors. *Information Control*, 1960, 3, 60-67.

Bourne, C. P. & Ford, D. J. A study of methods for systematically abbreviating English words and names. *Journal of the ACM*, 1961, 8(4), 538-552.

Bourne, C. P. Frequency and impact of spelling errors in bibliographic data bases. *Information Processing Management*, 1977, 13(1), 1-12.

Butler, A. K. & Frye, C. H. *PLANIT language reference manual*. Technical Memorandum TM-(L)-4422/002/01, Santa Monica: System Development Corporation, 1970.

Carroll, J. B., Davies, P., & Richman, B. *The American heritage word frequency book*. New York: Houghton Mifflin, 1971.

Damerau, F. J. A technique for computer detection and correction of spelling errors. *Communications of the ACM*, 1964, 7(3), 171-176.

Davidson, L. Retrieval of misspelled names in an airline's passenger record system. *Communications of the ACM*, 1962, 5(3), 169-171.

Faulk, R. D. An inductive approach to language translation. *Communications of the ACM*, 1964, 7(11), 647-653.

Feingold, S. L. *User's guide to PLANIT: Programming LANguage for Interactive Teaching*. Technical memorandum TM-3055/000/01, Santa Monica: System Development Corporation, 1966.

Fendt, W. E. *Bausteine in APL*. Projekt DV-B V 320 1/73, Heidelberg: Wissenschaftliches Institut fur Rehabilitation und berufliche Bildung, 1974.

Feurzeig, W. A. *Computer systems for teaching complex concepts*. Report no. 1742, Cambridge: Bolt Beranek and Newman, 1969.

Feurzeig, W. A. New instructional potentials of information technology. *IEEE Transactions on Human Factors in Electronics*, 1967, 8(2), 84-88.

Glantz, H. T. On the recognition of information with a digital computer. *Journal of the ACM*, 1957, 4, 178-188.

Gleason, H. A. *An introduction to descriptive linguistics.* New York: Holt Rinehart and Winston, 1961.

Hall, P. A. & Dowling, G. R. Approximate string matching. *Computing Surveys*, 1980, 12(4), 381-402.

Hanna, P. R. *Phoneme-grapheme correspondences as cues to spelling improvement*. Washington D.C.: U.S. Department of Health Education and Welfare, 1966.

Hewes, W. L. & Stowe, K. H. Information retrieval by proper name. *Data Processing Magazine*, June, 1965, 18-22.

Honeywell Information Systems *NATAL II language specification manual*. ZB07 revision 0.0, 1981.

Hutchinson, L. I. *Standard handbook for secretaries.* New York: McGraw-Hill, 1956.

IBM Corporation *IBM 1500 coursewriter II, author's guide.* Y26-1580-0,San Jose: Author, 1968.

Jackson, M. Mnemonics. *Datamation*, April, 1967, 26-28.

Jordan, J. (ed.) *A handbook for terrible spellers*. New York: Inovation Press, 1963.

Kashyap, R. L. & Oommen B. J. An effective algorithm for string correction using generalized edit distances. *Information Sciences*, 1981, 23, 123-142; 201-217.

Knuth, D. E. *The art of computer programming, volume 3: sorting and searching*. Reading: Addison-Wesley, 1973.

Kucera, H. & Francis, W. N. Computational analysis of present-day American English. Brown University Press, 1967.

Lowrance, R. & Wagner, R. A. An extension of the string-to-string correction problem. *Journal of the ACM*, 1975, 22(2), 177-183.

Masters, H. V. *A study of spelling errors*. Unpublished doctoral dissertation, University of Iowa, 1927.

Mor, M. & Fraenkel, A. S. A hash code method for detecting and correcting spelling errors. *Communications of the*

ACM, 1982, 25(12), 935-938.

Morgan, H. L. Spelling correction in systems programs. *Communications of the ACM*, 1970, 13(2), 90-94.

Muth, F. E. & Tharp, A. L. Correcting human error in alphanumeric terminal input. *Information Processing Management*, 1977, 13(6), 329-337.

Paice, C. D. *Information retrieval and the computer*. London: Macdonald and Jane's Publishers, 1977.

Peterson, J. L. *Computer Programs for Spelling Correction*. New York: Springer-Verlag, 1980.

Peuchot, M. *Instruction Module Generator (IMOGENE) : general information manual* Paris: IBM Corporation (France), 1975.

Riseman, E. M. & Hanson, A. R. A contextual post-processing system for error-correction using binary n-grams. *IEEE Transactions on Computers*, 1974, c-23(5), 480-493.

Romaniuk, E. & Schienbein, R. *Coursewriter II function guide to the DERS CAI system*. Division of Educational Research Services manual, Edmonton: University of Alberta, 1973.

Szanser, A. J. Error-correcting methods in natural language processing. *Information Processing 68*, IFIP proceedings, 1969, 1412-1416.

Szanser, A. J. *Automatic error correction in natural texts*. NPL Report COM 53, Teddington(U.K.): National Physical Laboratory, 1971.

Szanser, A. J. *Automatic error correction in natural texts - supplement*. NPL Report COM 63, Teddington(U.K.): National Physical Laboratory, 1973a.

Szanser, A. J. Bracketing technique in elastic matching. *The Computer Journal*, 1973b, 16(2), 132-134.

Symonds, M. *Computer detection of misspelled words*. ERB-843, Ottawa: National Research Council of Canada, 1970.

Tenczar, P. J. & Golden, W. M. *Spelling, word, and concept recognition*. Computer-based Education Research Laboratory Report X-55, Urbana: University of Illinois, 1972.

Thorelli, L. E. Automatic correction of errors in text. *BIT 2*, 1962, 2, 45-62.

Wagner, R. A. & Fischer, M. J. The string-to-string

correction problem. *Journal of the ACM*, 1974, 21(1), 168-178.

Westrom, M. L. *NATAL-74 specification manual*. NRC 14245, Ottawa: National Research Council of Canada, 1974.

Wijk, A. *Rules of pronounciation for the English language.* London: Oxford Press, 1966.

Zipf, G. K. *Human behavior and the principle of least effort; an introduction to human ecology.* Cambridge: Addison-Wesley, 1949.

Appendix A. Word Data Lists

Blair Word Data

absor/bent	absorbant
absor/ption	absorbtion
accommodat/e	accomodate
acquiesc/e	aquiese
analy/ze	analize
antarctic/	antartic
asinin/e	assinine
assist/ance	assistence
auxiliar/y	auxillary
banana/	bananna
bankrupt/cy	bankrupcy
brethren/	bretheren
brit/ain	britian
buoy/ancy	bouyancy
categor/y	catagorey
chauffeur/	chauffuer
chimney/s	chimnies
coliseum/	colosium
colossal/	collosal
commit/ment	committment
committee/	commitee
conced/e	consede
conscien/tious	conscientous
consensus/	concensus

controver/sy	controvercy
corrugat/ed	corrigated
cynic/al	synical
deuce/	duece
develop/	devellope
digni/tary	dignatary
disappoint	disapoint
drastic/ally	drasticly
ecsta/sy	ecstacy
embarrass/	embarass
exaggerat/e	exagerate
exist/ence	existance
exten/sion	extention
february/	febuary
fi/ery	firey
filipin/os	philipinoes
flammable/	flamable
forthright/	fortright
forty/	fourty
fulfill/	fullfil
gnaw/ing	knawing
govern/ment	goverment
gramma/r	grammer
heartrending/	heartrendering
hemorrhag/e	hemorrage
hindrance/	hinderence
hygien/e	hygeine

idiosyncra/sy	idiocyncracy
incens/e	insense
incidental/ly	incidently
infallib/le	infalable
inoculat/e	innoculate
insist/ence	insistance
interced/e	intersede
interfer/ed	interferred
jeopard/ize	jeprodise
kimono/	kimona
licens/e	lisence
liquef/y	liquify
maint/enance	maintainance
manag/ement	managment
maneuver/	manuveur
mortgag/ed	mortgauged
nickel/	nickle
ninetynin/th	nintynineth
nowadays/	nowdays
occasion/ally	ocassionaly
occurr/ence	occurence
pamphlet/	phamplet
permissi/ble	permissable
persever/ance	perseverence
persua/de	pursuade
philippine/s	phillipines
pittsburgh/	pittsburg

plagiaris/m	plaigarism
playwright/	playwrite
prairie/	prarie
preced/ing	preceeding
precipice/	presipice
prefer/able	preferrable
presumptuous/	presumptous
privilege/	privelege
propel/ler	propellor
psycholog/ical	psycological
public/ly	publically
pursue/r	persuer
questionnaire/	questionaire
recipient/	resipient
relevan/t	revelent
renown/	renoun
repel/	repell
rhapsody/	raphsody
rhododendron/	rhododrendon
rhubarb/	ruhbarb
rhythm/	rythm
sacrileg/ious	sacreligious
safe/ty	safty
scissors/	sissers
se/ize	sieze
separat/e	seperate
shepherd/	sheperd

similar/	similiar
sincer/ity	sincerety
souvenir/	souviner
specimen/	speciment
sui/ng	sueing
surreptitious/	sureptitous
transfer/able	transferrable
unparallel/ed	unparalelled
us/age	useage
vegetable/	vegatable
wednesday/	wedensday
weird/	wierd

Damerau Word Data

abdel/	abdul
aggressi/on	agression
algiers/	algeirs
anniversary/	aniversary
antarctic/	antartic
barabashev/	barbashev barbashov
chiang/	chaing
colombo/	columbo
communis/t	cmmunist
congressm/an	conressman
consumer/	cosumer
dalai/	dalal
foreign/	foriegn

frontier/	fronier
ghosh/	gosh
grotewohl/	grotewahl
guerilla/	guerrila
independen/ce	independance
jodrell/	jodreu
khinzemens/	khimzamene
khrushchev/	khrushcev krushchev khrushev
kozlov/	koslov
kuibyshev/	kuibishev
longju/	longnu
mohammed/	mohamed
negotiat/ion	negociation
philippin/es	phillipines
phongsaly/	phonsaly
pittsburgh/	pittsburg
plebiscite/	plebescite
prague/	prage
research/	rearch
satellite/	sattelite
sirimavo/	sirimauo
southeast/	souteast
suslov/	suscov
thankhek/	thankek
ulbright/	ulbrigt
vecerni/	vercerni

vientianne/	vientanne
visit/	vist

Masters Word Data

accommodat/e	accomodate
accommodat/ion	accomodation accomidation
accompan/ying	accompaning
accrue/d	accrude acrude acrued
accustom/ed	accustom acustomed
acknowledg/ment	acknowledgement
acquaint/ance	acquaintence acquantance
adjourn/ed	adjourn ajourned
alumni/	alumnae
amiable/	aimiable aimeable
anniversary/	aniversary anniversery
annoyance/	anoyance annoiance
anticipat/e	antisipate antisapate antispate
anticipat/ing	antisipating anticapating
anticipat/ion	antisipation antisapation
apolog/y	appology

apparent/	apparant aparent
apparent/ly	apparantly apperently
appetite/	appitite appatite apetite
appropriat/e	appropiate apropriate
approximat/ely	approximatly aproximately
arrangement/s	arrangments arangements
ascertain/	assertain asertain accertain acertain
attach/ing	attatching attacting
attorney/s	attornies
bankrupt/cy	bankrupcy bankrupsy
beneficia/l	benefical benificial
bore/d	board bord
bungalow/	bungalo
buried/	burried barried varied
cancel/ed	cancel
cancel/lation	cancelation
canvass/	canvas
catalog/ues	catalogue catlogue catologues

cease/	seize sease
chemist/ry	chemestry
collateral/	calateral
commit/ted	commited committee
committee/s	commities committes
communicat/e	comunicate community
comparative/ly	comparitively
competent/	compitent competant
complet/ing	completeing
conceiv/e	concieve conseve
concept/ion	conseption
confirm/ation	conformation confermation
conscienc/e	conscious concience
conscious/	concious consious
consisten/t	consistant
continu/ous	continous continious
controvers/y	controversey controvercy
convenient/ly	conviently
correspond/ence	correspondance correspondents
coun/sel	council
curiosity/	curiousity

curious/	courious qurious
damage/d	damage
debit/	debt debet
decided/ly	dicidely
deem/	deam
definite/ly	definately definatly
delegat/es	deligates
delinquent/	delinquint deliguent
deny/	denie denigh
despair/	dispair dispare
determin/ed	determine detirmined
dining/	dinning
disappoint/	disapoint dissapoint dessapoint dissappoint
disappoint/ment	dissapointment dissappointment
discretion/	discression disgression
divine/	devine
dormitory/	dormatory dormotory
drop/ped	dropt droped
duly/	duely dully
dying/	dieing

edition/	addition
efficien/cy	effeciency
elementary/	elimentary elementry
eliminat/e	illiminate elliminate
employ/ees	employies employes
enem/ies	enimies enemys
entrance/	enterance enterence
equip/ped	equiped equipt
ere/	err
esteem/ed	esteem estimed
exist/ence	existance
exist/s	exist exsist
exquisite/	exqusite exquiset exquisit
fascinat/ing	facinating fasinating
folly/	fally folley
fundamental/	fundemental fundimental
galvanize/d	galvanize galvinized
genius/	genious geneous
gorge/ous	georgeous gorgas

grateful/	greatful
grippe/	grip
guarantee/d	guarantee gauranteed
guardian/	gaurdian guardien gardian
imitat/ion	immitation imatation
immense/ly	immensly imensely immencely
incidentally/	incidently
inconvenien/ce	inconvience
inconvenien/ced	inconvienced inconvenience
indefinite/	indefinate
infinit/e	infinate
innocent/	inocent
itemiz/ed	itemize itimized
kindergarten/	kindergarden
laborator/y	labratory
literal/ly	literaly litterally
magnificent/	magnificant magnigicient
material/ly	materialy
mathematic/s	mathamatics mathmatics
melancholy/	melconcholy meloncaly
mortgage/	morgage

myster/ious	misterious
myster/y	mistery mystry
necessar/ily	necessarilly necessarially
occasional/ly	occassionally occasionly
opportunit/ies	opportunity oppurtunities
ordinar/ily	ordinarilly
original/ly	origionally orginally
pamphlet/	phamplet pamplet pamphalet
pamphlet/s	phamplets pamplets
partial/	parcial parcel
perceiv/e	percieve
phase/	faze fase
physician/	physican position physcian
pos/sess	posess posses
prefer/red	prefered
prejudic/e	predjudice predudice
principle/s	principals
prior/	prier pryor
privileg/e	priviledge privelege

procedure/	proceedure
questionnaire/	questionaire
rating/	rateing raiting
receipt/s	reciepts receits recipts
receive/r	reciever
reckon/	recon reccon
recommend/	reccommend recomend
recommend/ation	recomendation reccommendation
recommend/ations	reccomendations recomendations reccommendations
recommend/ed	reccommended recomended recommend
recommend/ing	reccommending recomending reccomending
refer/red	refered
refer/ring	refering
regret/ted	regreted
rememb/rance	rememberance rememberence
representati/ves	represenatives representives representitives
restaurant/	resturant restaraunt
ridiculous/	rediculous
romantic/	romatic

satisfact/orily	satisfactorly satisfactorally
scandal/	scandle scandel
schedule/d	schedule
seize/d	siezed ceased
solemn/	solomn solmn
sororit/y	sororiety sarrority
specific/ally	specificly spacifically
specimen/	speciman
specimen/s	specimans
strenuous/	streneous strenous
sufficient/ly	sufficently suficiently
supplement/	suplement supliment
temporar/y	temperary tempory
thorough/	through thourough
thorough/ly	throughly thourghly thouroughly
tonnage/	tonage
traged/y	tradegy tradgedy
transfer/red	transfered
undoubted/ly	undoubtly undoubtably

unfortunate/ly	unfortunatly unfortionately
unusual/ly	unusual unusally usually usualy
vacanc/y	vacency vancancy
viol/ence	violets violance
virtu/e	vertue virture
visib/le	visable
voucher/	vulture vouture

Nesbit Word Data

accept/ed	acceped acceppted acepted aspect eccepted exccepted excepted exepted
ache/	aaek ace ack acke eak
actor/	acter aktter
almost/	allmost olmoest olmost
already/	allready alrday olrede olredy
although/	allthough

	alltough
	athough
alway/s	alaway
	alway
	alwes
	awes
among/	amond
	amoug
	amoung
	amuge
	amung
angel/	angle
any/	enoy
anything/	anething
	enething
	enithing
apron/	aprine
arriv/e	arive
arriv/ed	arived
	drive
assignment/	asingment
	assiment
	assinghment
	assinment
author/	arthor
	ather
	athor
	aurther
	auther
	authur
	autor
	awither
away/	uay
because/	becauce
	becaus
	becaus
beggar/	baggar
	bagger
	beager
	beger
	begger

	bugger pegger
bell/y	belley bley
beneath/	beneith
blossom/s	blossems
bluff/	bulf
book/let	booklit
bought/	boght
brok/en	brokon
buil/d	biuld
burglar/	bargaler berglar bruglar brugller buglar burgaler burgeler burgglar burglor burgular burler
buy/	by
calculat/or	calclater
calv/es	calfs
capit/al	captial
captain/	captian captin
cattle/	cattel
cedar/	ceader ceddar ceder cetar cittar seader seator seatter sedder

	seder seeder seedor setar
cent/ral	centeral
cent/re	center centure centurn
certain/	centen certan certein certin sertain surtain
cheap/	cheep
chemi/stry	cemestery
chemi/cal	camecal cemacul cemecal cemical cemicle
chickenpox/	chickenpocks
chose/n	chousen cosen
christmas/	cristmas
cinderella/	cinderela cinderrela sinderela sindrelue
circ/le	cicle circal circel curcal
combin/e	combind combined comfine conbine
concert/	consert
consent/	concent concint

contain/s	contane contanes contans
continu/e	continie continu contiu contiune
cousin/s	cousons
crowd/	crowed
cruel/	crool crual crule
curtain/	certen certian curtane
daily/	dayly
dance/r	danncor
dazzl/e	dazzel
decid/e	decied deside dicide
deserv/e	desurve
destr/oy	distor
difficult/	diffacllt diffuclt
does/	dase dose dous
doesnt/	dosent
donor/	doanor donar donear doner donner donnor dooner
dozen/	dosen

drawer/s	draws
drove/	drov
earl/y	eraly erly
eighty/	eagity eigty etei
electr/ic	electrec eletric
endless/	endlees
engage/	agage engajde engange ingade ingage ingaged
erupt/ion	eraption eropsion eroption erupshon
eskimo/s	eskamos
exercis/e	excercise exersis exersize exrsise
fairy/	faiy fary
favor/ite	faverit favorit
field/	feild
fir/e	frie
fire/d	fride
fo/ur	for fore
friend/	freind
fur/	fir

furni/ture	furnitchure
garden/	gardin
geolog/y	gedgily geoligy geologey gologay
gingerbread/	gigrbred
grammar/	crammer gramar gramer grammer grammor gramor
gretel/	gretl grettel
half/	haff
hansel/	hancel hansl hasel
hav/ing	haveing
head/	haid haid hed
he/ar	here
herd/	heard hered hurd
hour/	huor
includ/e	enclude
instead/	insted
instrument/	insterment
interest/ed	intrested
junior/	juinor junier
ladder/	latter

less/on	leson
liber/ty	libaty libraty
limb/	lim
liqui/d	liqud liqued
magic/	magce
major/	mager magor manger
man/y	manay meny
maple/	mapel
massag/e	masach masaga masage masoge masosh masuage masuage mesash misage musoshe musouge
maybe/	manbe
mayor/	maiar maire major mangor marrior marrir marror
measle/s	mesels
militar/y	miletary millytary
mirror/	mearor mere mieeor mirrier mirrior

model/	modle motle
moist/	most
motor/	modor motar moter motter mottor
motor/ized	motorised
muddle/d	mudald muddeld mudduld mudled muttled
mutton/	meten moten muten mutten
necessar/y	necasary neccisary nessacary nessacery nessasry nesseary nessecery nesseray
necklace/	neclace
nerv/ous	nerves
nickel/	nickle
ninety/	nighty nintey nity
north/ern	northeren
ocean/	oacen otion
omit/	omeit
operator/	operater opereator oporator

	oporeator opperator opporator opreator
pajama/s	pejamas
pas/s	parss
pas/sed	paste
pass/age	pasag pasage pasige
pas/te	paset past
percent/	persent
perfume/	perfoum perfum pervum pirfum purfum purfume
period/	pariod peareaid pearid perid peried perieod periode perioid pired
phon/e	phown
piece/	peice
pimples/	pimpls pimppal pimppleas pimpules
planet/	pannet planit
pleas/e	plaes
poe/m	poum
poe/t	poety

	poit pouet powit
polar/	palor poler poller polor polor
possib/le	posible
prince/	pries
princes/s	prines
priz/e	prise
probab/ly	probibly
purpos/e	perpus
puzzl/e	puzzel
qua/ke	qauk quacke
qui/te	quit
rac/ing	raceing
raze/	raies rais raise rase rays
read/y	rede redte redy
real/ly	realy
regain/	regan regane
rifle/	rifal rifel rifiel
right/	write
rough/	rofe roff

	ruff
	rug
ruf/f	roffe
	rouf
	roufe
	rough
	ruf
	rufe
sailor/	sailar
	sailer
	sallor
	salor
scen/e	sene
sentenc/e	sentance
	sentince
settl/ed	seteld
	seteled
	setled
	setteld
	setteled
	settld
	settles
several/	sevaral
	severel
	sevral
	sevrall
shin/y	shiney
shovel/	shovle
shr/ank	shrak
	shranke
simpl/e	simmpal
	simpaill
size/	sies
slic/e	slise
soccer/	socer
sour/	sawr
	sror
squ/are	saer

squirrel/	squirle
steer/	stear stire
stoop/	stup
stor/y	storie
str/ike	srake striek
success/	succees succes succese sucess sucsses susses
successiv/e	secseveve secsive sicseciv sicsesof sicsesove succesive succesof successuf succesuve succeufe suckseof sucseccof sucsesive sucsesof sucsesofe sucsesseve sucsessive sucssevive sugcesive sugesive sussisive
sure/	sur
surviv/ing	serviving surfiving surrving surving surviveing suviving
ta/le	taile
televis/ion	telivision tellevision

term/s	turns
throat/	throte
through/	thour throgh throu
thumb/	thoum
too/	to
toward/s	towords
towel/	taule towle
troll/	trole
tunnel/	tunel
turtle/	turtel
val/ley	valliy
view/	veiw vieu
visitor/	visator visiter visittor vister vistor
weigh/	wiegh
which/	whitch wich
whit/e	whit whitte wiht wite
who/se	whoes
wise/	wase wies
with/	whith
wors/t	worsest

wrist/	rist
writ/ing	righting wirting wrighting writting

Appendix B. Words Deleted From Dictionary

afire	message
afield	modal
ahead	modeled
air	our
airplanes	paced
airy	paz
anesthetics	peace
anne	peas
annie	radii
awe	raise
chang	raising
dale	rally
daley	ray
daly	relay
delay	rely
dose	salem
doze	seen
drive	seine
ehre	sen
er	shanty
erie	shines
fer	shore
fez	sighs
fiery	sin
fife	skeptical
georges	stir
gorilla	tel
hues	tier
jib	two
lassen	whiz

Appendix C. Pascal Source Code for the Algorithms Tested in Experiment I

```
const   length = 80;

type    chars = 0..length;
        string = record
                 line : ARRAY[chars] OF CHAR;
                 len : chars
                 end;
        stringlist = array[1..length] of string;

function stest(arg1,arg2 : string) : boolean ;
var i : integer;
begin
stest := true;
if arg1.len <> arg2.len
    then stest := false
    else for i := 1 to arg1.len do
        if  arg1.line[i] <> arg2.line[i]
            then stest := FALSE;
end;

procedure delete (var str : string; pos : integer);
var i : integer;
begin
for i := pos to str.len do str.line[i] := str.line[i+1];
str.len := str.len-1;
end;

procedure truncate (var str : string; pos : integer);
var i : integer;
begin
if str.len>pos then
    begin
    str.len := pos;
    for i := pos downto 1 do
    if str.line[i]=' ' then str.len := i-1;
    end;
end;

procedure delchar
(var str : string; c : char; pos : integer);
var i : integer;
begin
for i := str.len downto pos do
    if str.line[i]=c then delete(str,i);
end;

procedure deladj (var str : string);
var i : integer;
begin
for i := str.len downto 2 do
    if str.line[i]=str.line[i-1] then delete(str,i);
```

```
end;

procedure BLAIR (a,b : string; var r : integer);
var tlen : integer;
    procedure code (var str : string);
    type vector = array [1..length] of integer;
    string9 = packed array[1..9] of char;
    var i,j,k,diff,maxpos : integer;
        charscore : vector;
        wstr1,wstr2 :string9;
    begin
    for i := 1 to str.len do
        case str.line[i] of
        'd','j','q','x':    charscore[i] := 0;
        'b','f','k','m',
        'v','w','z':        charscore[i] := 1;
        'g','y':            charscore[i] := 2;
        'n','p','t':        charscore[i] := 3;
        'o','r','u':        charscore[i] := 4;
        'a','c','h','l',
        's':                charscore[i] := 5;
        'i':                charscore[i] := 6;
        'e':                charscore[i] := 7;
        otherwise write('BAD STRING');
        end;
    wstr1 := '024556667';
    wstr2 := '134556677';
    i := 1;
    j := str.len;
    while i<=j do
        begin
        if i<9 then k := i else k := 9;
        charscore[i] := charscore[i] + (ord(wstr1[k])-48);
        if i<j then
            charscore[j]:=charscore[j]+(ord(wstr2[k])-48);
        i := i+1;
        j := j-1;
        end;
    diff := str.len - tlen;
    for j := 1 to diff do
        begin
        maxpos := 1;
        for i := 1 to str.len do
            if charscore[i]>=charscore[maxpos]
                then maxpos := i;
        delete(str,maxpos);
        for i := maxpos to str.len do
            charscore[i] := charscore[i+1];
        end;
    end;

begin
tlen := 4;
code(a);
```

```
code(b);
if stest(a,b) then r := 1 else r := 0;
end;

procedure DAMERAU (a,b : string; var r : integer);
var m,n,diff,errorcount,firsterror,lasterror : integer;
    procedure match (length : integer);
    var i : integer;
    begin
    errorcount := 0;
    firsterror := 0;
    lasterror  := 0;
    for i := 1 to length do
        begin
        if a.line[i] <> b.line[i] then
            begin
            errorcount := errorcount + 1;
            if errorcount=1
                then firsterror := i
                else lasterror := i;
            end;
        end;
    end;

begin
m := a.len;
n := b.len;
diff := m - n;
if (diff < -1) or (diff > 1)
    then r := 0
    else case diff of
        0:  begin
            match(m);
            if errorcount < 3
                then
                    case errorcount of
                    0,1:    r := 1;
                    2:      if (firsterror = lasterror-1)
                            and
                     (a.line[firsterror] = b.line[lasterror])
                            and
                     (b.line[firsterror] = a.line[lasterror])
                                then r := 1
                                else r := 0;
                    end
                else r := 0;
            end;
        -1: begin
            match(m);
            if errorcount = 0
                then r := 1
                else
                    begin
                    delete(b,firsterror);
```

```
                match(m);
                if errorcount = 0
                    then r := 1
                    else r := 0;
                end;
        end;
    +1: begin
        match(n);
        if errorcount = 0
            then r := 1
            else
                begin
                delete(a,firsterror);
                match(n);
                if errorcount = 0
                    then r := 1
                    else r := 0;
                end;
        end;
    end;
end;

procedure SYMONDS (a,b : string; var r : integer);
var i,j,asize,bsize : integer;
    alist,blist : stringlist;
    procedure parse
    (str : string; var codelist : stringlist;
        var size : integer);
    var h,i,j,k,newsize  : integer;
        procedure nplicate(n : integer);
        var p : integer;
        begin
        newsize := n * size;
        for p := 1 to n-1 do
            for k := 1 to size do
                begin
                for h:=1 to codelist[k].len do
                    codelist[k+size*p].line[h] :=
                        codelist[k].line[h];
                codelist[k+size*p].len := codelist[k].len;
                end;
        k := 1;
        end;
    procedure catlist(c : char);
    begin
    j := k;
    while j < k+size do
        begin
        codelist[j].len := codelist[j].len + 1;
        codelist[j].line[codelist[j].len] := c;
        j := j + 1;
        end;
    k := j;
    if k>newsize then
```

```
        begin
        size := newsize;
        k := 1;
        end;
    end;
begin
i := 1;
k := 1;
size := 1;
newsize := 1;
codelist[1].len := 0;
str.line[str.len+1] := ' ';
while i <= str.len do
    if (i>1) and (str.line[i]=str.line[i-1])
        then i := i + 1
        else
            case str.line[i] of
            'b':    begin
                    catlist('B');
                    i := i+1;
                    end;
            'c':
                case str.line[i+1] of
                'h':    begin
                        nplicate(2);
                        catlist('s');
                        catlist('C');
                        i := i+2;
                        end;
                'k':    begin
                        catlist('K');
                        i := i+2;
                        end;
                'q':    begin
                        catlist('Q');
                        i := i+2;
                        end
                otherwise
                        begin
                        nplicate(2);
                        catlist('S');
                        catlist('K');
                        i := i+1;
                        end
                end;
            'd':
                case str.line[i+1] of
                'g':    begin
                        nplicate(2);
                        catlist('J');
                        catlist('G');
                        i := i+2;
                        end
                otherwise
```

```
                begin
                catlist('D');
                i := i+1;
                end
        end;
    'f':    begin
            catlist('F');
            i := i+1;
            end;
    'g':
        case str.line[i+1] of
        'h': if str.line[i+2]='t'
            then
                begin
                nplicate(2);
                catlist('F');
                k := 1;
                catlist('T');
                catlist('T');
                i := i+3;
                end
            else
                begin
                nplicate(2);
                catlist('F');
                catlist('G');
                i := i+2;
                end;
        'n':    begin
                catlist('N');
                i := i+2;
                end
        otherwise
                begin
                nplicate(2);
                catlist('J');
                catlist('G');
                i := i+1;
                end
        end;
    'h':    begin
            catlist('H');
            i := i+1;
            end;
    'j':    begin
            catlist('J');
            i := i+1;
            end;
    'k':    if str.line[i+1]='n'
                then
                    begin
                    catlist('N');
                    i := i+2;
                    end
```

```
                    else
                        begin
                        catlist('K');
                        i := i+1;
                        end;
        'l':    begin
                catlist('L');
                i := i+1;
                end;
        'm':    begin
                catlist('M');
                i := i+1;
                end;
        'n':    begin
                catlist('N');
                i := i+1;
                end;
        'p':    if str.line[i+1]='h'
                    then
                        begin
                        catlist('F');
                        i := i+2;
                        end
                    else
                        begin
                        catlist('P');
                        i := i+1;
                        end;
        'q':    begin
                catlist('Q');
                i := i+1;
                end;
        'r':    if str.line[i+1]='h'
                    then
                        begin
                        catlist('R');
                        i := i+2;
                        end
                    else
                        begin
                        catlist('R');
                        i := i+1;
                        end;
        's':
            case str.line[i+1] of
            'c':    if (str.line[i+2]='i') or
                    (str.line[i+2]='e')
                        then
                            begin
                            nplicate(2);
                            catlist('S');
                            catlist('s');
                            i := i+3;
                            end
```

```
                        else
                            begin
                            nplicate(2);
                            catlist('S');
                            catlist('Z');
                            i := i+1;
                            end;
            'h':    begin
                    catlist('s');
                    i := i+2;
                    end
            otherwise
                    begin
                    nplicate(2);
                    catlist('S');
                    catlist('Z');
                    i := i+1;
                    end
            end;
        't':
            case str.line[i+1] of
            'c':    if str.line[i+2]='h'
                        then
                            begin
                            nplicate(3);
                            catlist('s');
                            catlist('C');
                            catlist('K');
                            i := i+3;
                            end
                        else
                            begin
                            catlist('T');
                            i := i+1;
                            end;
            'i':    if str.line[i+2]='o'
                        then
                            begin
                            nplicate(2);
                            catlist('C');
                            catlist('s');
                            i := i+3;
                            end
                        else
                            begin
                            catlist('T');
                            i := i+1;
                            end;
            'h':    begin
                    catlist('t');
                    i := i+2;
                    end
            otherwise
                    begin
```

```
                        catlist('T');
                        i := i+1;
                        end
                end;
            'w':    if i<>str.len
                        then
                            if str.line[i+1]='h'
                                    then
                                    begin
                                    catlist('W');
                                    i := i+2;
                                    end
                                    else
                                    begin
                                    catlist('W');
                                    i := i+1;
                                    end
                        else i := i+1;
            'x':    begin
                    nplicate(2);
                    catlist('K');
                    k := 1;
                    catlist('S');
                    catlist('G');
                    size := size div 2;
                    k := size+1;
                    catlist('Z');
                    i := i+1;
                    end;
            'v':    begin
                    catlist('V');
                    i := i+1;
                    end;
            'y':    if i=1
                        then
                            begin
                            catlist('Y');
                            i := i+1;
                            end
                        else i:=i+1;
            'z':    begin
                    catlist('Z');
                    i := i+1;
                    end
            otherwise i:=i+1
            end;
end;
begin
parse(a,alist,asize);
parse(b,blist,bsize);
i := 0;
r := 0;
while (r=0) and (i<asize) do
    begin
```

```
    i := i+1;
    j := 0;
    while (r=0) and (j<bsize) do
        begin
        j := j+1;
        if stest(alist[i],blist[j]) then r:=1;
        end;
    end;
end;

procedure DAMERAU-SYMONDS (a,b : string; var r : integer);
begin
DAMERAU(a,b,r);
if r=0 then SYMONDS(a,b,r);
end;

procedure SOUNDEX (a,b : string; var r : integer);
    procedure code (var str : string);
    var i : integer;
    begin
    for i := 2 to str.len do
        case str.line[i] of
        'a','e','i','o',
        'u','h','w','y':    str.line[i] := 'A';
        'b','f','p','v':    str.line[i] := 'B';
        'c','g','j','k',
        'q','s','x','z':    str.line[i] := 'C';
        'd','t':            str.line[i] := 'D';
        'l':                str.line[i] := 'L';
        'm','n':            str.line[i] := 'M';
        'r':                str.line[i] := 'R';
        otherwise writeln('BAD STRING ');
        end;
    delchar(str,'A',2);
    deladj(str);
    truncate(str,4);
    end;
begin
code(a);
code(b);
if stest(a,b) then r := 1 else r := 0;
end;

procedure PLANIT (a,b : string; var r : integer);
    procedure code (var str : string);
    var i : integer;
    begin
    for i := 1 to str.len do
        case str.line[i] of
        'a','e','i','o',
        'u','y':            str.line[i] := 'A';
        'b','f','p','v':    str.line[i] := 'B';
        'c','g','j','k',
        'q','s','x','z':    str.line[i] := 'C';
```

```
        'd','t':            str.line[i] := 'D';
        'h','w':            str.line[i] := 'H';
        'l':                str.line[i] := 'L';
        'm','n':            str.line[i] := 'M';
        'r':                str.line[i] := 'R';
        otherwise write('BAD STRING ');
        end;
    delchar(str,'H',2);
    deladj(str);
    delchar(str,'A',1);
    end;
begin
code(a);
code(b);
if stest(a,b) then r := 1 else r := 0;
end;

procedure ROOF-SBYC-STRING (a,b : string; var r : integer);
    var m,n,i,j,k : integer;
        rtemp : real;
        coin : array[1..length,1..length] of real;
    function max(arg1,arg2 : integer): integer;
        begin
        if arg1 > arg2 then max := arg1 else max := arg2;
        end;
    procedure roof;
        begin
        if (m=1) or (n=1)
            then
                for i := 1 to m do
                    for j := 1 to n do
                        coin[i,j] :=
                            coin[i,j] * (1-abs(i/m-j/n))
            else
                for i := 1 to m do
                    for j := 1 to n do
                        coin[i,j] :=
            coin[i,j] * (1-abs((i-1)/(m-1)-(j-1)/(n-1)))
        end;
    procedure sbyc;
        var sel,baki : integer;
            flag : boolean;
    begin
    for i := 1 to m do
        begin
        sel := 1;
        for j := 1 to n do
            if coin[i,j] > coin[i,sel]
                then
                    begin
                    flag := false;
                    for baki := i-1 downto 1 do
                        if coin[baki,j]<>0
                            then flag := true;
```

```
                    if flag=false then sel := j;
                    end;
        for j := 1 to n do
        if j<>sel then coin[i,j] := 0;
        end;
    end;
    procedure stri;
    begin
    rtemp := 0;
    for i := 1 to m do
        for j := 1 to n do
            if coin[i,j]<>0
                then
                    begin
                    k := 0;
                    while (i+k<=m)
                    and (j+k<=n)
                    and (coin[i+k,j+k]<>0) do k := k+1;
                    rtemp := rtemp + coin[i,j] * k;
                    end;
    k := max(m,n);
    k := (k**2+k) div 2;         (* normalizing divisor *)
    r := round(100*(rtemp/k));
    end;

begin
m := a.len;
n := b.len;
for i := 1 to m do
    for j := 1 to n do
        if a.line[i]=b.line[j]
            then coin[i,j] := 1
            else coin[i,j] := 0;
roof;
sbyc;
stri;
end;

procedure WAGNER (a,b : string; var r : integer);
var i,j,m,n,min,diff : integer;
    dist,idcost,sml,sub,ins,del : real;
    d : array [-1..20,-1..20] of real;
    function subcost : real;
        begin
        if a.line[i] = b.line[j]
            then subcost := 0.0
            else subcost := 0.7;
        end;
begin
idcost := 0.5;
r := 0;
m := a.len;
n := b.len;
if m < n then min := m else min := n;
```

```
diff := abs(m-n);
d[0,0] := 0;
for i := 1 to m do d[i,0] := d[i-1,0]+idcost;
for j := 1 to n do d[0,j] := d[0,j-1]+idcost;
for i := 1 to m do
    for j := 1 to n do
        begin
        sub := d[i-1,j-1] + subcost;
        ins := d[i-1,j]   + idcost;
        if ins<sub then sml := ins else sml := sub;
        del := d[i,j-1]   + idcost;
        if del<sml then sml := del;
        d[i,j] := sml;
        end;
(*  The worst case edit distance is:

            min(m,n) * subcost
                    +
            diff(m,n) * idcost

    which can become a normalizing factor to get a metric
    between 0 and 100.
*)
dist := d[m,n];
r := round(100*(1-dist/(min*subcost+diff*idcost)));
end;

procedure HALL (a,b : string; var r : integer);
var i,j,m,n,min,diff : integer;
    inf,dist,idcost,tracost,sml,sub,ins,del,tra : real;
    d : array [-1..20,-1..20] of real;
    function subcost(i,j : integer) : real;
    begin
    if a.line[i] = b.line[j]
        then subcost := 0.0
        else subcost := 0.7;
    end;

begin
idcost := 0.5;
tracost := 0.2;
r := 0;
m := a.len;
n := b.len;
if m < n then min := m else min := n;
diff := abs(m-n);
inf := m * idcost + n * idcost;
d[0,0] := 0;
d[-1,-1] := 0;
d[0,-1] := inf;
d[-1,0] := inf;
for i := 1 to m do
    begin
    d[i,0] := d[i-1,0] + idcost;
```

```
    d[i,-1] := inf;
    end;
for j := 1 to n do
    begin
    d[0,j] := d[0,j-1] + idcost;
    d[-1,j] := inf;
    end;
for i := 1 to m do
    for j := 1 to n do
        begin
        sub := d[i-1,j-1] + subcost(i,j);
        ins := d[i-1,j]   + idcost;
        if ins<sub then sml := ins else sml := sub;
        del := d[i,j-1]   + idcost;
        if del<sml then sml := del;
        if (j>1) and (i>1)
            then
                begin
                tra := d[i-2,j-2] + subcost(i-1,j) +
                subcost(i,j-1) + tracost;
                if tra<sml then sml := tra;
                end;
        d[i,j] := sml;
        end;
dist := d[m,n];
r := round(100 * (1 - dist/(min * 0.7 + diff * idcost)));
end;
```

Appendix D. Results of Experiments I and III

The following table shows type I and type II error values recorded for the algorithms tested in experiments I and III with all four word data lists. It is structured as a matrix with word data lists comprising the horizontal axis and algorithms comprising the vertical axis. Each experimental event is represented in the table by five values in the following manner:

(frequency type I error)	(frequency type II error)
(proportion type I error)	(mean frequency type II error)
	(proportion type II error)

The proportion type I error and mean type II error frequency were calculated as described on page 68 and were graphed in figures 15 through 18.

Recalling that the non-binary algorithms were programmed to return an integer between 0 and 100 inclusive, note that each threshold set on that returned value is represented here as an independent algorithm. Threshold settings are indicated in the leftmost column underneath the name of the algorithm to which they apply. Comparisons returning a value greater than or equal to the threshold were defined as resulting in a match. To save space, the data reported here range from the highest threshold setting at which no type I error were found to the lowest threshold at which no type II errors were found.

To take an example, with a threshold setting of 75, HN7 produced the following results when tested with the Nesbit word data:

84	1483
0.160	6.962
	3.21605E-04

This means that 84 word-misspelling pairs failed to yield a value greater than or equal to 75, and that these 84 constituted 16% of such pairs in the Nesbit word data. Of all comparisons between words from the Nesbit word data and words from the dictionary, 1483 failed to yield a value less than 75. These constituted about .03% of all such comparisons and represent a mean of about 7 matches in the dictionary for every word in the word data list.

Function	Blair		Damerau		Masters		Nesbit	
BLAIR	28	357	13	98	109	649	308	391
	0.239	3.051	0.295	2.390	0.341	3.626	0.588	1.836
		1.40923E-04		1.10369E-04		1.67430E-04		8.47928E-05
DAMERAU	31	18	8	6	91	80	245	427
	0.265	0.154	0.182	0.146	0.284	0.447	0.468	2.005
		7.10537E-06		6.75730E-06		2.06386E-05		9.25998E-05
SYMONDS	20	252	20	66	69	421	143	1381
	0.171	2.154	0.455	1.610	0.216	2.352	0.273	6.484
		9.94751E-05		7.43303E-05		1.08610E-04		2.99486E-04
DAMERAU-SYMONDS	7	439	4	138	43	905	102	3626
	0.060	3.752	0.091	3.366	0.134	5.056	0.195	17.023
		1.73292E-04		1.55418E-04		2.33474E-04		7.86339E-04
SOUNDEX	6	2421	9	575	38	3746	89	4275
	0.051	20.692	0.205	14.024	0.119	20.927	0.170	20.070
		9.55672E-04		6.47574E-04		9.66400E-04		9.27082E-04
PLANIT	9	2345	13	857	61	3242	88	8000
	0.077	20.043	0.295	20.902	0.191	18.112	0.168	37.559
		9.25671E-04		9.65167E-04		8.36377E-04		1.73489E-03
ROOF-SBYC-STRING								
5	0	901706	0	328047	0	1382188	0	1904787
	0.000	7706.889	0.000	8001.146	0.000	7721.721	0.000	8942.662
		3.55942E-01		3.69452E-01		3.56579E-01		4.13075E-01
6	0	593464	0	219284	0	920841	1	1368177
	0.000	5072.342	0.000	5348.390	0.000	5144.363	0.002	6423.366
		2.34265E-01		2.46961E-01		2.37560E-01		2.96705E-01
7	0	383351	0	145974	0	608213	1	1012636
	0.000	3276.504	0.000	3560.342	0.000	3397.838	0.002	4754.160
		1.51325E-01		1.64398E-01		1.56908E-01		2.19602E-01
8	0	244707	0	96117	0	396325	1	695173
	0.000	2091.513	0.000	2344.317	0.000	2214.106	0.002	3263.723
		9.65963E-02		1.08249E-01		1.02245E-01		1.50756E-01
9	0	154268	0	62173	0	258305	3	497721
	0.000	1318.530	0.000	1516.415	0.000	1443.045	0.006	2336.718

		6.08961E-02		7.00202E-02		6.66380E-02		1.07936E-01
10	0	104304	0	43333	0	178289	5	372166
	0.000	891.487	0.000	1056.902	0.000	996.028	0.010	1747.258
		4.11732E-02		4.88023E-02		4.59953E-02		8.07084E-02
11	0	67461	0	27740	0	118550	6	250455
	0.000	576.590	0.000	676.585	0.000	662.291	0.011	1175.845
		2.66297E-02		3.12412E-02		3.05838E-02		5.43140E-02
12	0	47814	0	19510	0	85794	6	194191
	0.000	408.667	0.000	475.854	0.000	479.296	0.011	911.695
		1.88742E-02		2.19725E-02		2.21333E-02		4.21125E-02
13	0	33585	0	13598	1	62971	10	144595
	0.000	287.051	0.000	331.659	0.003	351.793	0.019	678.850
		1.32574E-02		1.53143E-02		1.62454E-02		3.13571E-02
14	0	22939	0	8938	3	44216	13	102987
	0.000	196.060	0.000	218.000	0.009	247.017	0.025	483.507
		9.05500E-03		1.00661E-02		1.14069E-02		2.23339E-02
15	0	16553	0	6496	3	32129	19	79116
	0.000	141.479	0.000	158.439	0.009	179.492	0.036	371.437
		6.53417E-03		7.31590E-03		8.28870E-03		1.71572E-02
16	0	12067	0	4699	4	24541	23	62829
	0.000	103.137	0.000	114.610	0.013	137.101	0.044	294.972
		4.76336E-03		5.29209E-03		6.33113E-03		1.36252E-02
17	0	9521	0	3685	6	19721	31	52511
	0.000	81.376	0.000	89.878	0.019	110.173	0.059	246.531
		3.75834E-03		4.15011E-03		5.08766E-03		1.13876E-02
18	1	7338	0	2843	6	15225	41	40442
	0.009	62.718	0.000	69.341	0.019	85.056	0.078	189.869
		2.89662E-03		3.20183E-03		3.92777E-03		8.77031E-03
19	2	5697	1	2082	6	11854	48	31495
	0.017	48.692	0.023	50.780	0.019	66.223	0.092	147.864
		2.24885E-03		2.34478E-03		3.05812E-03		6.83005E-03
20	2	4289	1	1678	6	9238	60	25105
	0.017	36.658	0.023	40.927	0.019	51.609	0.115	117.864
		1.69305E-03		1.88979E-03		2.38324E-03		5.44431E-03

21	2	3223	1	1247	6	7068	64	17579
	0.017	27.547	0.023	30.415	0.019	39.486	0.122	82.531
		1.27226E-03		1.40439E-03		1.82342E-03		3.81221E-03
22	2	2575	3	991	8	5717	70	15525
	0.017	22.009	0.068	24.171	0.025	31.939	0.134	72.887
		1.01646E-03		1.11608E-03		1.47488E-03		3.36677E-03
23	3	2122	3	859	10	4874	75	13519
	0.026	18.137	0.068	20.951	0.031	27.229	0.143	63.469
		8.37644E-04		9.67420E-04		1.25740E-03		2.93175E-03
24	4	1727	3	705	12	4018	84	11296
	0.034	14.761	0.068	17.195	0.038	22.447	0.160	53.033
		6.81720E-04		7.93982E-04		1.03657E-03		2.44967E-03
25	5	1418	3	544	13	3392	92	9709
	0.043	12.120	0.068	13.268	0.041	18.950	0.176	45.582
		5.59745E-04		6.12662E-04		8.75075E-04		2.10551E-03
26	6	1163	3	437	17	2859	100	8064
	0.051	9.940	0.068	10.659	0.053	15.972	0.191	37.859
		4.59086E-04		4.92156E-04		7.37570E-04		1.74877E-03
27	6	970	4	361	20	2457	114	7003
	0.051	8.291	0.091	8.805	0.063	13.726	0.218	32.878
		3.82900E-04		4.06564E-04		6.33862E-04		1.51868E-03
28	7	717	4	269	25	1979	127	5865
	0.060	6.128	0.091	6.561	0.078	11.056	0.242	27.535
		2.83030E-04		3.02952E-04		5.10546E-04		1.27189E-03
29	7	624	4	229	30	1794	140	5227
	0.060	5.333	0.091	5.585	0.094	10.022	0.267	24.540
		2.46319E-04		2.57904E-04		4.62820E-04		1.13353E-03
30	13	516	4	187	35	1528	146	4464
	0.111	4.410	0.091	4.561	0.109	8.536	0.279	20.958
		2.03687E-04		2.10602E-04		3.94196E-04		9.68069E-04
31	18	424	6	148	44	1172	159	3003
	0.154	3.624	0.136	3.610	0.138	6.547	0.303	14.099
		1.67371E-04		1.66680E-04		3.02355E-04		6.51235E-04
32	19	365	6	140	47	1037	172	2740
	0.162	3.120	0.136	3.415	0.147	5.793	0.328	12.864

		1.44081E-04		1.57670E-04		2.67527E-04		5.94200E-04
33	23	317	7	124	54	904	184	2362
	0.197	2.709	0.159	3.024	0.169	5.050	0.351	11.089
		1.25133E-04		1.39651E-04		2.33216E-04		5.12227E-04
34	25	263	9	108	59	742	198	2014
	0.214	2.248	0.205	2.634	0.184	4.145	0.378	9.455
		1.03817E-04		1.21631E-04		1.91423E-04		4.36759E-04
35	26	237	10	105	63	686	205	1832
	0.222	2.026	0.227	2.561	0.197	3.832	0.391	8.601
		9.35540E-05		1.18253E-04		1.76976E-04		3.97290E-04
36	31	214	11	91	70	611	211	1745
	0.265	1.829	0.250	2.220	0.219	3.413	0.403	8.192
		8.44749E-05		1.02486E-04		1.57627E-04		3.78423E-04
37	33	174	12	74	74	522	218	1525
	0.282	1.487	0.273	1.805	0.231	2.916	0.416	7.160
		6.86852E-05		8.33400E-05		1.34667E-04		3.30714E-04
38	38	162	12	70	75	470	228	1426
	0.325	1.385	0.273	1.707	0.234	2.626	0.435	6.695
		6.39483E-05		7.88351E-05		1.21251E-04		3.09244E-04
39	39	144	16	62	84	424	252	1255
	0.333	1.231	0.364	1.512	0.262	2.369	0.481	5.892
		5.68429E-05		6.98254E-05		1.09384E-04		2.72161E-04
40	41	126	16	55	91	401	255	1196
	0.350	1.077	0.364	1.341	0.284	2.240	0.487	5.615
		4.97376E-05		6.19419E-05		1.03451E-04		2.59366E-04
41	43	76	17	42	98	288	273	753
	0.368	0.650	0.386	1.024	0.306	1.609	0.521	3.535
		3.00004E-05		4.73011E-05		7.42988E-05		1.63297E-04
42	50	68	17	40	110	263	304	724
	0.427	0.581	0.386	0.976	0.344	1.469	0.580	3.399
		2.68425E-05		4.50486E-05		6.78492E-05		1.57008E-04
43	52	58	18	30	113	236	310	696
	0.444	0.496	0.409	0.732	0.353	1.318	0.592	3.268
		2.28951E-05		3.37865E-05		6.08837E-05		1.50936E-04

44	55	47	23	26	127	191	339	574
	0.470	0.402	0.523	0.634	0.397	1.067	0.647	2.695
		1.85529E-05		2.92816E-05		4.92745E-05		1.24478E-04
45	64	46	26	25	147	180	350	556
	0.547	0.393	0.591	0.610	0.459	1.006	0.668	2.610
		1.81582E-05		2.81554E-05		4.64367E-05		1.20575E-04
46	72	41	26	20	163	160	371	513
	0.615	0.350	0.591	0.488	0.509	0.894	0.708	2.408
		1.61844E-05		2.25243E-05		4.12771E-05		1.11250E-04
47	80	40	26	19	176	158	376	510
	0.684	0.342	0.591	0.463	0.550	0.883	0.718	2.394
		1.57897E-05		2.13981E-05		4.07611E-05		1.10599E-04
48	86	31	29	17	194	145	394	466
	0.735	0.265	0.659	0.415	0.606	0.810	0.752	2.188
		1.22370E-05		1.91457E-05		3.74074E-05		1.01057E-04
49	87	20	29	14	195	118	398	355
	0.744	0.171	0.659	0.341	0.609	0.659	0.760	1.667
		7.89485E-06		1.57670E-05		3.04419E-05		7.69858E-05
50	92	19	31	13	206	92	405	322
	0.786	0.162	0.705	0.317	0.644	0.514	0.773	1.512
		7.50011E-06		1.46408E-05		2.37343E-05		6.98294E-05
51	96	16	33	12	219	85	429	285
	0.821	0.137	0.750	0.293	0.684	0.475	0.819	1.338
		6.31588E-06		1.35146E-05		2.19285E-05		6.18055E-05
52	97	15	35	8	226	76	437	276
	0.829	0.128	0.795	0.195	0.706	0.425	0.834	1.296
		5.92114E-06		9.00973E-06		1.96066E-05		5.98537E-05
53	100	15	36	7	235	72	453	270
	0.855	0.128	0.818	0.171	0.734	0.402	0.865	1.268
		5.92114E-06		7.88351E-06		1.85747E-05		5.85526E-05
54	104	15	37	7	242	69	457	264
	0.889	0.128	0.841	0.171	0.756	0.385	0.872	1.239
		5.92114E-06		7.88351E-06		1.78008E-05		5.72514E-05
55	106	10	38	7	248	50	458	243
	0.906	0.085	0.864	0.171	0.775	0.279	0.874	1.141

		3.94742E-06		7.88351E-06		1.28991E-05		5.26973E-05
56	109	10	39	7	254	50	474	239
	0.932	0.085	0.886	0.171	0.794	0.279	0.905	1.122
		3.94742E-06		7.88351E-06		1.28991E-05		5.18299E-05
57	109	9	40	6	264	46	477	225
	0.932	0.077	0.909	0.146	0.825	0.257	0.910	1.056
		3.55268E-06		6.75730E-06		1.18672E-05		4.87938E-05
58	111	6	40	6	272	45	488	218
	0.949	0.051	0.909	0.146	0.850	0.251	0.931	1.023
		2.36845E-06		6.75730E-06		1.16092E-05		4.72758E-05
59	111	6	40	6	277	44	489	216
	0.949	0.051	0.909	0.146	0.866	0.246	0.933	1.014
		2.36845E-06		6.75730E-06		1.13512E-05		4.68421E-05
60	111	6	41	6	280	44	490	213
	0.949	0.051	0.932	0.146	0.875	0.246	0.935	1.000
		2.36845E-06		6.75730E-06		1.13512E-05		4.61915E-05
61	111	6	41	6	285	34	493	122
	0.949	0.051	0.932	0.146	0.891	0.190	0.941	0.573
		2.36845E-06		6.75730E-06		8.77138E-06		2.64571E-05
62	111	5	42	6	289	32	500	118
	0.949	0.043	0.955	0.146	0.903	0.179	0.954	0.554
		1.97371E-06		6.75730E-06		8.25542E-06		2.55896E-05
63	111	4	42	5	293	30	502	104
	0.949	0.034	0.955	0.122	0.916	0.168	0.958	0.488
		1.57897E-06		5.63108E-06		7.73946E-06		2.25536E-05
64	111	4	42	5	295	28	502	104
	0.949	0.034	0.955	0.122	0.922	0.156	0.958	0.488
		1.57897E-06		5.63108E-06		7.22349E-06		2.25536E-05
65	112	4	42	4	298	26	503	104
	0.957	0.034	0.955	0.098	0.931	0.145	0.960	0.488
		1.57897E-06		4.50486E-06		6.70753E-06		2.25536E-05
66	112	4	42	4	300	26	504	103
	0.957	0.034	0.955	0.098	0.938	0.145	0.962	0.484
		1.57897E-06		4.50486E-06		6.70753E-06		2.23367E-05

67	112	4	42	4	300	26	504	103
	0.957	0.034	0.955	0.098	0.938	0.145	0.962	0.484
		1.57897E-06		4.50486E-06		6.70753E-06		2.23367E-05
68	112	2	43	3	303	13	507	36
	0.957	0.017	0.977	0.073	0.947	0.073	0.968	0.169
		7.89485E-07		3.37865E-06		3.35376E-06		7.80701E-06
69	112	2	43	3	303	13	507	36
	0.957	0.017	0.977	0.073	0.947	0.073	0.968	0.169
		7.89485E-07		3.37865E-06		3.35376E-06		7.80701E-06
70	112	2	43	3	303	13	507	36
	0.957	0.017	0.977	0.073	0.947	0.073	0.968	0.169
		7.89485E-07		3.37865E-06		3.35376E-06		7.80701E-06
71	113	2	43	3	305	13	507	36
	0.966	0.017	0.977	0.073	0.953	0.073	0.968	0.169
		7.89485E-07		3.37865E-06		3.35376E-06		7.80701E-06
72	114	2	43	3	308	6	515	7
	0.974	0.017	0.977	0.073	0.962	0.034	0.983	0.033
		7.89485E-07		3.37865E-06		1.54789E-06		1.51803E-06
73	114	2	43	3	308	6	515	7
	0.974	0.017	0.977	0.073	0.962	0.034	0.983	0.033
		7.89485E-07		3.37865E-06		1.54789E-06		1.51803E-06
74	114	1	43	3	308	4	515	7
	0.974	0.009	0.977	0.073	0.962	0.022	0.983	0.033
		3.94743E-07		3.37865E-06		1.03193E-06		1.51803E-06
75	114	1	43	2	311	4	518	7
	0.974	0.009	0.977	0.049	0.972	0.022	0.989	0.033
		3.94743E-07		2.25243E-06		1.03193E-06		1.51803E-06
76	115	1	43	2	312	1	523	0
	0.983	0.009	0.977	0.049	0.975	0.006	0.998	0.000
		3.94743E-07		2.25243E-06		2.57982E-07		0.00000E+00
77	115	1	43	2	312	0	523	0
	0.983	0.009	0.977	0.049	0.975	0.000	0.998	0.000
		3.94743E-07		2.25243E-06		0.00000E+00		0.00000E+00
78	116	1	43	2	314	0	523	0
	0.991	0.009	0.977	0.049	0.981	0.000	0.998	0.000

		3.94743E-07		2.25243E-06		0.00000E+00		0.00000E+00
79	116	1	43	0	314	0	524	0
	0.991	0.009	0.977	0.000	0.981	0.000	1.000	0.000
		3.94743E-07		0.00000E+00		0.00000E+00		0.00000E+00
80	117	1	44	0	318	0	524	0
	1.000	0.009	1.000	0.000	0.994	0.000	1.000	0.000
		3.94743E-07		0.00000E+00		0.00000E+00		0.00000E+00
81	117	0	44	0	319	0	524	0
	1.000	0.000	1.000	0.000	0.997	0.000	1.000	0.000
		0.00000E+00		0.00000E+00		0.00000E+00		0.00000E+00
WAGNER								
15	0	1532222	0	504324	0	2448727	0	2728870
	0.000	13095.914	0.000	12300.585	0.000	13680.039	0.000	12811.597
		6.04833E-01		5.67978E-01		6.31727E-01		5.91787E-01
16	0	1439356	0	469562	0	2316326	1	2527782
	0.000	12302.188	0.000	11452.731	0.000	12940.369	0.002	11867.521
		5.68175E-01		5.28828E-01		5.97570E-01		5.48178E-01
17	0	1353566	0	438558	0	2187554	1	2306587
	0.000	11568.940	0.000	10696.536	0.000	12220.972	0.002	10829.047
		5.34310E-01		4.93911E-01		5.64349E-01		5.00210E-01
18	0	1284376	0	413222	0	2065646	1	2119235
	0.000	10977.572	0.000	10078.585	0.000	11539.922	0.002	9949.460
		5.06998E-01		4.65377E-01		5.32899E-01		4.59580E-01
19	0	1189675	0	381431	0	1912376	1	1956762
	0.000	10168.162	0.000	9303.195	0.000	10683.665	0.002	9186.676
		4.69615E-01		4.29574E-01		4.93358E-01		4.24346E-01
20	0	1094377	0	356936	0	1772877	1	1848602
	0.000	9353.649	0.000	8705.756	0.000	9904.341	0.002	8678.883
		4.31997E-01		4.01987E-01		4.57370E-01		4.00890E-01
21	0	1000131	0	325619	0	1640074	1	1711552
	0.000	8548.128	0.000	7941.927	0.000	9162.425	0.002	8035.456
		3.94794E-01		3.66717E-01		4.23109E-01		3.71170E-01
22	0	923871	0	296797	0	1511491	1	1574812

	0.000	7896.333	0.000	7238.951	0.000	8444.084	0.002	7393.483
		3.64691E-01		3.34258E-01		3.89937E-01		3.41516E-01
23	0	847663	0	269710	0	1393046	1	1465363
	0.000	7244.983	0.000	6578.292	0.000	7782.380	0.002	6879.639
		3.34609E-01		3.03752E-01		3.59381E-01		3.17781E-01
24	0	750891	0	236309	0	1231062	1	1301038
	0.000	6417.872	0.000	5763.634	0.000	6877.441	0.002	6108.160
		2.96409E-01		2.66135E-01		3.17592E-01		2.82145E-01
25	0	680774	0	217083	0	1119902	1	1201818
	0.000	5818.581	0.000	5294.708	0.000	6256.436	0.002	5642.338
		2.68730E-01		2.44482E-01		2.88914E-01		2.60628E-01
26	0	591550	0	188511	0	981983	1	1094582
	0.000	5055.983	0.000	4597.829	0.000	5485.938	0.002	5138.883
		2.33510E-01		2.12304E-01		2.53334E-01		2.37373E-01
27	0	533760	0	168910	0	881791	1	993242
	0.000	4562.051	0.000	4119.756	0.000	4926.207	0.002	4663.108
		2.10698E-01		1.90229E-01		2.27486E-01		2.15396E-01
28	0	477524	0	149327	0	787539	1	881737
	0.000	4081.402	0.000	3642.122	0.000	4399.659	0.002	4139.610
		1.88499E-01		1.68174E-01		2.03171E-01		1.91215E-01
29	0	418074	0	131837	0	704301	1	801665
	0.000	3573.282	0.000	3215.537	0.000	3934.643	0.002	3763.686
		1.65032E-01		1.48477E-01		1.81697E-01		1.73850E-01
30	0	375408	0	119123	0	636370	2	722808
	0.000	3208.615	0.000	2905.439	0.000	3555.140	0.004	3393.465
		1.48189E-01		1.34158E-01		1.64172E-01		1.56749E-01
31	0	328385	0	104016	0	561575	2	650925
	0.000	2806.709	0.000	2536.976	0.000	3137.291	0.004	3055.986
		1.29628E-01		1.17145E-01		1.44876E-01		1.41161E-01
32	0	280693	0	89038	0	491945	3	553443
	0.000	2399.085	0.000	2171.658	0.000	2748.296	0.006	2598.324
		1.10801E-01		1.00276E-01		1.26913E-01		1.20020E-01
33	0	248149	0	78005	0	435983	3	513095
	0.000	2120.932	0.000	1902.561	0.000	2435.659	0.006	2408.897
		9.79550E-02		8.78505E-02		1.12476E-01		1.11271E-01

34	0	209461	0	65329	0	370678	3	423658
	0.000	1790.265	0.000	1593.390	0.000	2070.827	0.006	1989.005
		8.26832E-02		7.35746E-02		9.56282E-02		9.18751E-02
35	0	187331	0	58847	0	328397	3	399627
	0.000	1601.120	0.000	1435.293	0.000	1834.620	0.006	1876.183
		7.39475E-02		6.62744E-02		8.47205E-02		8.66637E-02
36	0	162617	0	50252	0	286722	3	337487
	0.000	1389.889	0.000	1225.659	0.000	1601.799	0.006	1584.446
		6.41918E-02		5.65946E-02		7.39691E-02		7.31879E-02
37	0	140604	0	43969	0	250564	3	307513
	0.000	1201.744	0.000	1072.415	0.000	1399.799	0.006	1443.723
		5.55024E-02		4.95186E-02		6.46410E-02		6.66877E-02
38	0	119521	0	37451	1	213629	4	259803
	0.000	1021.547	0.000	913.439	0.003	1193.458	0.008	1219.732
		4.71800E-02		4.21779E-02		5.51124E-02		5.63412E-02
39	0	100772	0	31091	1	182326	4	223916
	0.000	861.299	0.000	758.317	0.003	1018.581	0.008	1051.249
		3.97790E-02		3.50152E-02		4.70368E-02		4.85587E-02
40	0	87252	0	26753	1	158916	6	201687
	0.000	745.744	0.000	652.512	0.003	887.799	0.011	946.887
		3.44421E-02		3.01297E-02		4.09974E-02		4.37381E-02
41	0	73200	0	21876	2	136478	9	167946
	0.000	625.641	0.000	533.561	0.006	762.447	0.017	788.479
		2.88952E-02		2.46371E-02		3.52089E-02		3.64210E-02
42	0	63199	0	18955	2	119082	10	159576
	0.000	540.162	0.000	462.317	0.006	665.263	0.019	749.183
		2.49473E-02		2.13474E-02		3.07210E-02		3.46059E-02
43	0	50720	0	15075	3	99481	15	126465
	0.000	433.504	0.000	367.683	0.009	555.760	0.029	593.732
		2.00213E-02		1.69777E-02		2.56643E-02		2.74254E-02
44	0	46612	0	13616	3	90571	16	115134
	0.000	398.393	0.000	332.098	0.009	505.983	0.031	540.535
		1.83997E-02		1.53346E-02		2.33657E-02		2.49681E-02
45	0	39931	0	11559	3	78220	17	105768

	0.000	341.291	0.000	281.927	0.009	436.983	0.032	496.563
		1.57625E-02		1.30179E-02		2.01793E-02		2.29370E-02
46	0	32736	0	9107	3	63150	20	79278
	0.000	279.795	0.000	222.122	0.009	352.793	0.038	372.197
		1.29223E-02		1.02565E-02		1.62916E-02		1.71923E-02
47	0	27929	0	7875	3	54647	20	74041
	0.000	238.709	0.000	192.073	0.009	305.290	0.038	347.610
		1.10248E-02		8.86895E-03		1.40979E-02		1.60566E-02
48	0	21055	0	6319	3	43518	21	57874
	0.000	179.957	0.000	154.122	0.009	243.117	0.040	271.709
		8.31130E-03		7.11656E-03		1.12269E-02		1.25506E-02
49	0	18456	0	5391	3	38119	21	54224
	0.000	157.744	0.000	131.488	0.009	212.955	0.040	254.573
		7.28537E-03		6.07143E-03		9.83401E-03		1.17591E-02
50	0	15708	0	4580	3	32257	21	47213
	0.000	134.256	0.000	111.707	0.009	180.207	0.040	221.657
		6.20062E-03		5.15807E-03		8.32172E-03		1.02387E-02
51	0	14589	0	4290	4	29415	24	41557
	0.000	124.692	0.000	104.634	0.013	164.330	0.046	195.103
		5.75890E-03		4.83147E-03		7.58854E-03		9.01211E-03
52	0	11543	0	3550	5	24340	32	35677
	0.000	98.658	0.000	86.585	0.016	135.978	0.061	167.498
		4.55651E-03		3.99807E-03		6.27928E-03		7.73696E-03
53	0	9252	0	2779	5	20092	43	27551
	0.000	79.077	0.000	67.780	0.016	112.246	0.082	129.347
		3.65216E-03		3.12975E-03		5.18337E-03		5.97475E-03
54	0	7572	0	2188	5	16631	46	23830
	0.000	64.718	0.000	53.366	0.016	92.911	0.088	111.878
		2.98899E-03		2.46416E-03		4.29050E-03		5.16781E-03
55	0	6380	0	1706	5	13763	49	20213
	0.000	54.530	0.000	41.610	0.016	76.888	0.094	94.897
		2.51846E-03		1.92132E-03		3.55060E-03		4.38342E-03
56	0	5679	0	1464	6	11966	52	16034
	0.000	48.538	0.000	35.707	0.019	66.849	0.099	75.277
		2.24174E-03		1.64878E-03		3.08701E-03		3.47716E-03

57	0	4417	0	1112	6	9951	52	13787
	0.000	37.752	0.000	27.122	0.019	55.592	0.099	64.728
		1.74358E-03		1.25235E-03		2.56718E-03		2.98987E-03
58	0	3535	0	948	7	8284	54	12620
	0.000	30.214	0.000	23.122	0.022	46.279	0.103	59.249
		1.39541E-03		1.06765E-03		2.13712E-03		2.73679E-03
59	0	3250	0	870	7	7592	54	12032
	0.000	27.778	0.000	21.220	0.022	42.413	0.103	56.488
		1.28291E-03		9.79808E-04		1.95860E-03		2.60928E-03
60	0	2525	0	704	7	6234	56	10881
	0.000	21.581	0.000	17.171	0.022	34.827	0.107	51.085
		9.96725E-04		7.92856E-04		1.60826E-03		2.35967E-03
61	0	2106	0	570	8	4927	72	7375
	0.000	18.000	0.000	13.902	0.025	27.525	0.137	34.624
		8.31328E-04		6.41943E-04		1.27108E-03		1.59935E-03
62	0	1855	0	496	8	4151	72	6996
	0.000	15.855	0.000	12.098	0.025	23.190	0.137	32.845
		7.32247E-04		5.58603E-04		1.07088E-03		1.51716E-03
63	0	1352	0	398	10	3468	79	5871
	0.000	11.556	0.000	9.707	0.031	19.374	0.151	27.563
		5.33692E-04		4.48234E-04		8.94681E-04		1.27319E-03
64	0	1203	0	352	10	3054	81	5353
	0.000	10.282	0.000	8.585	0.031	17.061	0.155	25.131
		4.74875E-04		3.96428E-04		7.87877E-04		1.16086E-03
65	1	928	0	297	12	2508	105	3727
	0.009	7.932	0.000	7.244	0.038	14.011	0.200	17.498
		3.66321E-04		3.34486E-04		6.47019E-04		8.08242E-04
66	1	706	0	211	14	2147	111	2950
	0.009	6.034	0.000	5.146	0.044	11.994	0.212	13.850
		2.78688E-04		2.37632E-04		5.53887E-04		6.39741E-04
67	1	605	0	199	14	1912	112	2838
	0.009	5.171	0.000	4.854	0.044	10.682	0.214	13.324
		2.38819E-04		2.24117E-04		4.93261E-04		6.15453E-04
68	1	498	0	152	18	1389	124	2186

	0.009	4.256	0.000	3.707	0.056	7.760	0.237	10.263
		1.96582E-04		1.71185E-04		3.58337E-04		4.74059E-04
69	2	445	0	123	19	1202	130	1897
	0.017	3.803	0.000	3.000	0.059	6.715	0.248	8.906
		1.75660E-04		1.38525E-04		3.10094E-04		4.11386E-04
70	2	390	0	103	19	1081	130	1841
	0.017	3.333	0.000	2.512	0.059	6.039	0.248	8.643
		1.53950E-04		1.16000E-04		2.78878E-04		3.99242E-04
71	2	288	1	73	20	801	149	1230
	0.017	2.462	0.023	1.780	0.063	4.475	0.284	5.775
		1.13686E-04		8.22138E-05		2.06643E-04		2.66740E-04
72	7	193	1	53	23	634	172	968
	0.060	1.650	0.023	1.293	0.072	3.542	0.328	4.545
		7.61853E-05		5.96895E-05		1.63561E-04		2.09922E-04
73	7	153	1	48	24	546	175	931
	0.060	1.308	0.023	1.171	0.075	3.050	0.334	4.371
		6.03956E-05		5.40584E-05		1.40858E-04		2.01898E-04
74	7	138	1	46	24	518	176	931
	0.060	1.179	0.023	1.122	0.075	2.894	0.336	4.371
		5.44745E-05		5.18059E-05		1.33635E-04		2.01898E-04
75	7	101	2	32	30	329	204	591
	0.060	0.863	0.045	0.780	0.094	1.838	0.389	2.775
		3.98690E-05		3.60389E-05		8.48760E-05		1.28165E-04
76	9	84	2	28	32	272	223	426
	0.077	0.718	0.045	0.683	0.100	1.520	0.426	2.000
		3.31584E-05		3.15341E-05		7.01711E-05		9.23830E-05
77	10	79	3	26	35	254	243	402
	0.085	0.675	0.068	0.634	0.109	1.419	0.464	1.887
		3.11847E-05		2.92816E-05		6.55274E-05		8.71783E-05
78	11	69	3	24	36	214	243	398
	0.094	0.590	0.068	0.585	0.112	1.196	0.464	1.869
		2.72372E-05		2.70292E-05		5.52081E-05		8.63108E-05
79	13	41	3	15	41	161	258	302
	0.111	0.350	0.068	0.366	0.128	0.899	0.492	1.418
		1.61844E-05		1.68932E-05		4.15351E-05		6.54921E-05

80	13	37	3	12	41	153	260	302
	0.111	0.316	0.068	0.293	0.128	0.855	0.496	1.418
		1.46055E-05		1.35146E-05		3.94712E-05		6.54921E-05
81	18	24	7	11	54	109	305	189
	0.154	0.205	0.159	0.268	0.169	0.609	0.582	0.887
		9.47382E-06		1.23884E-05		2.81200E-05		4.09868E-05
82	18	22	8	9	56	84	313	143
	0.154	0.188	0.182	0.220	0.175	0.469	0.597	0.671
		8.68433E-06		1.01359E-05		2.16705E-05		3.10112E-05
83	27	15	10	8	72	71	322	141
	0.231	0.128	0.227	0.195	0.225	0.397	0.615	0.662
		5.92114E-06		9.00973E-06		1.83167E-05		3.05775E-05
84	33	13	13	7	81	44	347	91
	0.282	0.111	0.295	0.171	0.253	0.246	0.662	0.427
		5.13165E-06		7.88351E-06		1.13512E-05		1.97344E-05
85	35	13	14	5	85	41	348	90
	0.299	0.111	0.318	0.122	0.266	0.229	0.664	0.423
		5.13165E-06		5.63108E-06		1.05773E-05		1.95175E-05
86	37	6	17	4	93	31	371	53
	0.316	0.051	0.386	0.098	0.291	0.173	0.708	0.249
		2.36845E-06		4.50486E-06		7.99744E-06		1.14937E-05
87	48	4	18	4	108	21	408	38
	0.410	0.034	0.409	0.098	0.338	0.117	0.779	0.178
		1.57897E-06		4.50486E-06		5.41762E-06		8.24073E-06
88	50	4	19	4	109	19	408	38
	0.427	0.034	0.432	0.098	0.341	0.106	0.779	0.178
		1.57897E-06		4.50486E-06		4.90166E-06		8.24073E-06
89	61	2	21	1	128	10	452	9
	0.521	0.017	0.477	0.024	0.400	0.056	0.863	0.042
		7.89485E-07		1.12622E-06		2.57982E-06		1.95175E-06
90	78	1	23	1	165	4	488	1
	0.667	0.009	0.523	0.024	0.516	0.022	0.931	0.005
		3.94743E-07		1.12622E-06		1.03193E-06		2.16861E-07
91	82	1	24	1	185	4	490	1

	0.701	0.009	0.545	0.024	0.578	0.022	0.935	0.005
		3.94743E-07		1.12622E-06		1.03193E-06		2.16861E-07
92	89	1	31	0	210	4	510	0
	0.761	0.009	0.705	0.000	0.656	0.022	0.973	0.000
		3.94743E-07		0.00000E+00		1.03193E-06		0.00000E+00
93	96	0	36	0	248	2	518	0
	0.821	0.000	0.818	0.000	0.775	0.011	0.989	0.000
		0.00000E+00		0.00000E+00		5.15964E-07		0.00000E+00
94	112	0	44	0	292	0	524	0
	0.957	0.000	1.000	0.000	0.913	0.000	1.000	0.000
		0.00000E+00		0.00000E+00		0.00000E+00		0.00000E+00
HALL								
19	0	1320026	0	428447	0	2110837	0	2168223
	0.000	11282.273	0.000	10449.927	0.000	11792.386	0.000	10179.451
		5.21070E-01		4.82524E-01		5.44558E-01		4.70204E-01
20	0	1230659	0	402101	0	1972504	1	2056052
	0.000	10518.453	0.000	9807.342	0.000	11019.575	0.002	9652.826
		4.85793E-01		4.52853E-01		5.08870E-01		4.45878E-01
21	0	1136170	0	370711	0	1839957	1	1934168
	0.000	9710.854	0.000	9041.731	0.000	10279.090	0.002	9080.601
		4.48495E-01		4.17501E-01		4.74676E-01		4.19446E-01
22	0	1049578	0	340723	0	1701328	1	1787565
	0.000	8970.752	0.000	8310.317	0.000	9504.626	0.002	8392.324
		4.14313E-01		3.83728E-01		4.38912E-01		3.87654E-01
23	0	966560	0	312273	0	1575092	1	1665410
	0.000	8261.196	0.000	7616.415	0.000	8799.396	0.002	7818.826
		3.81542E-01		3.51687E-01		4.06345E-01		3.61163E-01
24	0	877335	0	279824	0	1421238	1	1530305
	0.000	7498.590	0.000	6824.976	0.000	7939.877	0.002	7184.530
		3.46321E-01		3.15142E-01		3.66654E-01		3.31864E-01
25	0	798698	0	256676	0	1295175	1	1419755
	0.000	6826.479	0.000	6260.390	0.000	7235.615	0.002	6665.517
		3.15280E-01		2.89073E-01		3.34132E-01		3.07890E-01
26	0	716385	0	230338	0	1166306	1	1308509

	0.000	6122.949	0.000	5618.000	0.000	6515.676	0.002	6143.235
		2.82788E-01		2.59410E-01		3.00886E-01		2.83765E-01
27	0	643833	0	207748	0	1050001	1	1190006
	0.000	5502.846	0.000	5067.024	0.000	5865.927	0.002	5586.883
		2.54148E-01		2.33969E-01		2.70881E-01		2.58066E-01
28	0	579380	0	186681	0	951293	1	1070578
	0.000	4951.966	0.000	4553.195	0.000	5314.486	0.002	5026.188
		2.28706E-01		2.10243E-01		2.45416E-01		2.32167E-01
29	0	518454	0	167045	0	862217	1	980113
	0.000	4431.231	0.000	4074.268	0.000	4816.855	0.002	4601.470
		2.04656E-01		1.88129E-01		2.22436E-01		2.12549E-01
30	0	466860	0	150175	0	779664	1	895755
	0.000	3990.256	0.000	3662.805	0.000	4355.665	0.002	4205.422
		1.84289E-01		1.69130E-01		2.01139E-01		1.94255E-01
31	0	408030	0	131053	0	686613	1	780557
	0.000	3487.436	0.000	3196.415	0.000	3835.827	0.002	3664.587
		1.61067E-01		1.47594E-01		1.77134E-01		1.69273E-01
32	0	355845	0	114690	0	604858	1	686981
	0.000	3041.410	0.000	2797.317	0.000	3379.095	0.002	3225.263
		1.40467E-01		1.29166E-01		1.56042E-01		1.48980E-01
33	0	316921	0	101749	0	540889	2	632719
	0.000	2708.727	0.000	2481.683	0.000	3021.726	0.004	2970.512
		1.25102E-01		1.14591E-01		1.39540E-01		1.37212E-01
34	0	274695	0	88643	0	466585	2	546976
	0.000	2347.821	0.000	2162.024	0.000	2606.620	0.004	2567.962
		1.08434E-01		9.98312E-02		1.20370E-01		1.18618E-01
35	0	244865	0	79137	0	413017	2	500569
	0.000	2092.863	0.000	1930.171	0.000	2307.357	0.004	2350.089
		9.66586E-02		8.91254E-02		1.06551E-01		1.08554E-01
36	0	209717	0	68108	0	359545	2	431349
	0.000	1792.453	0.000	1661.171	0.000	2008.631	0.004	2025.113
		8.27842E-02		7.67043E-02		9.27561E-02		9.35429E-02
37	0	184558	0	60614	0	318760	2	390479
	0.000	1577.419	0.000	1478.390	0.000	1780.782	0.004	1833.235
		7.28529E-02		6.82645E-02		8.22343E-02		8.46798E-02

38	0	159713	0	52377	0	275974	3	335851
	0.000	1365.068	0.000	1277.488	0.000	1541.754	0.006	1576.765
		6.30455E-02		5.89878E-02		7.11963E-02		7.28331E-02
39	0	133849	0	44395	0	235559	3	294526
	0.000	1144.009	0.000	1082.805	0.000	1315.972	0.006	1382.751
		5.28359E-02		4.99984E-02		6.07700E-02		6.38713E-02
40	0	115162	0	38129	0	203878	3	266637
	0.000	984.291	0.000	929.976	0.000	1138.983	0.006	1251.817
		4.54593E-02		4.29415E-02		5.25968E-02		5.78233E-02
41	0	99400	0	31464	1	176699	4	219942
	0.000	849.573	0.000	767.415	0.003	987.145	0.008	1032.592
		3.92374E-02		3.54353E-02		4.55851E-02		4.76969E-02
42	0	85911	0	27537	1	153994	4	208373
	0.000	734.282	0.000	671.634	0.003	860.302	0.008	978.277
		3.39127E-02		3.10126E-02		3.97277E-02		4.51881E-02
43	0	71198	0	22515	2	130895	9	166822
	0.000	608.530	0.000	549.146	0.006	731.257	0.017	783.202
		2.81049E-02		2.53568E-02		3.37685E-02		3.61773E-02
44	0	62866	0	19959	2	116069	10	150749
	0.000	537.316	0.000	486.805	0.006	648.430	0.019	707.742
		2.48159E-02		2.24781E-02		2.99437E-02		3.26916E-02
45	0	52164	0	16573	2	98281	10	134479
	0.000	445.846	0.000	404.220	0.006	549.056	0.019	631.357
		2.05913E-02		1.86648E-02		2.53547E-02		2.91633E-02
46	0	45076	0	14182	2	82907	10	111026
	0.000	385.265	0.000	345.902	0.006	463.168	0.019	521.249
		1.77934E-02		1.59720E-02		2.13885E-02		2.40773E-02
47	0	38613	0	12195	2	71052	12	102127
	0.000	330.026	0.000	297.439	0.006	396.939	0.023	479.469
		1.52422E-02		1.37342E-02		1.83301E-02		2.21474E-02
48	0	30339	0	10059	2	57909	14	81133
	0.000	259.308	0.000	245.341	0.006	323.514	0.027	380.906
		1.19761E-02		1.13286E-02		1.49395E-02		1.75946E-02
49	0	26071	0	8484	2	51021	14	72818

	0.000	222.829	0.000	206.927	0.006	285.034	0.027	341.869
		1.02913E-02		9.55482E-03		1.31625E-02		1.57914E-02
50	0	21687	0	7116	3	42234	15	62957
	0.000	185.359	0.000	173.561	0.009	235.944	0.029	295.573
		8.56078E-03		8.01415E-03		1.08956E-02		1.36529E-02
51	0	19216	0	6283	4	36904	20	52429
	0.000	164.239	0.000	153.244	0.013	206.168	0.038	246.146
		7.58537E-03		7.07602E-03		9.52056E-03		1.13698E-02
52	0	16034	0	5392	4	31409	20	47144
	0.000	137.043	0.000	131.512	0.013	175.469	0.038	221.333
		6.32930E-03		6.07256E-03		8.10295E-03		1.02237E-02
53	0	13184	0	4470	4	26637	28	38258
	0.000	112.684	0.000	109.024	0.013	148.810	0.053	179.615
		5.20429E-03		5.03419E-03		6.87186E-03		8.29668E-03
54	0	10974	0	3626	4	22321	29	34048
	0.000	93.795	0.000	88.439	0.013	124.698	0.055	159.850
		4.33190E-03		4.08366E-03		5.75841E-03		7.38370E-03
55	0	9040	0	2881	4	18232	36	28697
	0.000	77.265	0.000	70.268	0.013	101.855	0.069	134.728
		3.56847E-03		3.24463E-03		4.70353E-03		6.22327E-03
56	0	7888	0	2370	5	15403	39	22290
	0.000	67.419	0.000	57.805	0.016	86.050	0.074	104.648
		3.11373E-03		2.66913E-03		3.97369E-03		4.83384E-03
57	0	6084	0	1870	5	12672	39	19630
	0.000	52.000	0.000	45.610	0.016	70.793	0.074	92.160
		2.40161E-03		2.10602E-03		3.26915E-03		4.25699E-03
58	0	5154	0	1620	6	11000	40	18126
	0.000	44.051	0.000	39.512	0.019	61.453	0.076	85.099
		2.03450E-03		1.82447E-03		2.83780E-03		3.93083E-03
59	0	4325	0	1313	6	9468	41	14714
	0.000	36.966	0.000	32.024	0.019	52.894	0.078	69.080
		1.70726E-03		1.47872E-03		2.44257E-03		3.19090E-03
60	0	3537	0	1112	6	7871	42	13707
	0.000	30.231	0.000	27.122	0.019	43.972	0.080	64.352
		1.39620E-03		1.25235E-03		2.03058E-03		2.97252E-03

61	0	2997	0	930	7	6456	54	10176
	0.000	25.615	0.000	22.683	0.022	36.067	0.103	47.775
		1.18304E-03		1.04738E-03		1.66553E-03		2.20678E-03
62	0	2431	0	748	7	5250	57	9137
	0.000	20.778	0.000	18.244	0.022	29.330	0.109	42.897
		9.59619E-04		8.42410E-04		1.35440E-03		1.98146E-03
63	0	1930	0	633	9	4459	62	7666
	0.000	16.496	0.000	15.439	0.028	24.911	0.118	35.991
		7.61853E-04		7.12895E-04		1.15034E-03		1.66246E-03
64	0	1633	0	513	9	3788	64	6951
	0.000	13.957	0.000	12.512	0.028	21.162	0.122	32.634
		6.44615E-04		5.77749E-04		9.77235E-04		1.50740E-03
65	0	1344	0	430	11	3188	73	5316
	0.000	11.487	0.000	10.488	0.034	17.810	0.139	24.958
		5.30534E-04		4.84273E-04		8.22446E-04		1.15284E-03
66	0	1059	0	331	12	2690	77	4039
	0.000	9.051	0.000	8.073	0.038	15.028	0.147	18.962
		4.18032E-04		3.72778E-04		6.93971E-04		8.75903E-04
67	1	852	0	293	12	2304	78	3718
	0.009	7.282	0.000	7.146	0.038	12.872	0.149	17.455
		3.36321E-04		3.29981E-04		5.94390E-04		8.06291E-04
68	1	693	0	230	18	1744	88	2939
	0.009	5.923	0.000	5.610	0.056	9.743	0.168	13.798
		2.73557E-04		2.59030E-04		4.49920E-04		6.37356E-04
69	1	591	0	199	18	1504	100	2460
	0.009	5.051	0.000	4.854	0.056	8.402	0.191	11.549
		2.33293E-04		2.24117E-04		3.88005E-04		5.33479E-04
70	2	505	0	167	18	1305	104	2351
	0.017	4.316	0.000	4.073	0.056	7.291	0.198	11.038
		1.99345E-04		1.88078E-04		3.36666E-04		5.09841E-04
71	2	366	1	108	19	939	123	1535
	0.017	3.128	0.023	2.634	0.059	5.246	0.235	7.207
		1.44476E-04		1.21631E-04		2.42245E-04		3.32882E-04
72	3	277	1	84	20	750	131	1275

	0.026	2.368	0.023	2.049	0.063	4.190	0.250	5.986
		1.09344E-04		9.46022E-05		1.93486E-04		2.76498E-04
73	3	236	1	79	21	654	133	1245
	0.026	2.017	0.023	1.927	0.066	3.654	0.254	5.845
		9.31592E-05		8.89711E-05		1.68720E-04		2.69992E-04
74	3	208	1	71	21	591	135	1183
	0.026	1.778	0.023	1.732	0.066	3.302	0.258	5.554
		8.21064E-05		7.99613E-05		1.52467E-04		2.56547E-04
75	3	136	2	49	28	373	171	712
	0.026	1.162	0.045	1.195	0.087	2.084	0.326	3.343
		5.36850E-05		5.51846E-05		9.62272E-05		1.54405E-04
76	3	116	2	45	30	315	187	552
	0.026	0.991	0.045	1.098	0.094	1.760	0.357	2.592
		4.57901E-05		5.06797E-05		8.12643E-05		1.19707E-04
77	3	111	2	41	31	293	189	520
	0.026	0.949	0.045	1.000	0.097	1.637	0.361	2.441
		4.38164E-05		4.61749E-05		7.55887E-05		1.12768E-04
78	4	95	2	37	33	237	192	500
	0.034	0.812	0.045	0.902	0.103	1.324	0.366	2.347
		3.75005E-05		4.16700E-05		6.11417E-05		1.08431E-04
79	5	66	2	26	36	189	204	397
	0.043	0.564	0.045	0.634	0.112	1.056	0.389	1.864
		2.60530E-05		2.92816E-05		4.87586E-05		8.60940E-05
80	8	51	2	20	37	166	215	344
	0.068	0.436	0.045	0.488	0.116	0.927	0.410	1.615
		2.01319E-05		2.25243E-05		4.28250E-05		7.46003E-05
81	10	35	4	18	47	124	249	233
	0.085	0.299	0.091	0.439	0.147	0.693	0.475	1.094
		1.38160E-05		2.02719E-05		3.19898E-05		5.05287E-05
82	10	32	5	16	49	97	257	186
	0.085	0.274	0.114	0.390	0.153	0.542	0.490	0.873
		1.26318E-05		1.80195E-05		2.50242E-05		4.03362E-05
83	15	22	6	12	61	82	266	178
	0.128	0.188	0.136	0.293	0.191	0.458	0.508	0.836
		8.68433E-06		1.35146E-05		2.11545E-05		3.86013E-05

84	21 0.179	17 0.145 6.71062E-06	9 0.205	8 0.195 9.00973E-06	70 0.219	52 0.291 1.34151E-05	295 0.563	103 0.484 2.23367E-05
85	23 0.197	15 0.128 5.92114E-06	10 0.227	8 0.195 9.00973E-06	73 0.228	47 0.263 1.21251E-05	297 0.567	102 0.479 2.21199E-05
86	25 0.214	8 0.068 3.15794E-06	13 0.295	7 0.171 7.88351E-06	82 0.256	33 0.184 8.51340E-06	323 0.616	59 0.277 1.27948E-05
87	36 0.308	6 0.051 2.36845E-06	15 0.341	5 0.122 5.63108E-06	95 0.297	24 0.134 6.19157E-06	358 0.683	43 0.202 9.32504E-06
88	37 0.316	6 0.051 2.36845E-06	15 0.341	5 0.122 5.63108E-06	99 0.309	19 0.106 4.90166E-06	362 0.691	41 0.192 8.89132E-06
89	48 0.410	4 0.034 1.57897E-06	18 0.409	2 0.049 2.25243E-06	117 0.366	10 0.056 2.57982E-06	406 0.775	12 0.056 2.60234E-06
90	65 0.556	3 0.026 1.18423E-06	20 0.455	2 0.049 2.25243E-06	156 0.488	4 0.022 1.03193E-06	445 0.849	4 0.019 8.67446E-07
91	69 0.590	2 0.017 7.89485E-07	21 0.477	2 0.049 2.25243E-06	175 0.547	4 0.022 1.03193E-06	449 0.857	3 0.014 6.50584E-07
92	76 0.650	2 0.017 7.89485E-07	28 0.636	1 0.024 1.12622E-06	202 0.631	4 0.022 1.03193E-06	470 0.897	2 0.009 4.33723E-07
93	83 0.709	1 0.009 3.94743E-07	33 0.750	1 0.024 1.12622E-06	240 0.750	2 0.011 5.15964E-07	478 0.912	2 0.009 4.33723E-07
94	99 0.846	1 0.009 3.94743E-07	41 0.932	1 0.024 1.12622E-06	284 0.887	0 0.000 0.00000E+00	489 0.933	2 0.009 4.33723E-07
95	108	.0	41	0	303	0	502	0

	0.923	0.000	0.932	0.000	0.947	0.000	0.958	0.000
		0.00000E+00		0.00000E+00		0.00000E+00		0.00000E+00
HN7								
31	0	832557	0	267121	0	1357249	0	1402082
	0.000	7115.872	0.000	6515.146	0.000	7582.396	0.000	6582.544
		3.28646E-01		3.00836E-01		3.50146E-01		3.04057E-01
32	0	748236	0	238487	0	1223682	1	1271347
	0.000	6395.180	0.000	5816.756	0.000	6836.212	0.002	5968.765
		2.95361E-01		2.68588E-01		3.15688E-01		2.75706E-01
33	0	664541	0	211303	0	1091154	1	1150813
	0.000	5679.837	0.000	5153.732	0.000	6095.833	0.002	5402.878
		2.62323E-01		2.37973E-01		2.81498E-01		2.49567E-01
34	0	584128	0	185304	0	962649	1	1010065
	0.000	4992.547	0.000	4519.610	0.000	5377.927	0.002	4742.089
		2.30580E-01		2.08692E-01		2.48346E-01		2.19044E-01
35	0	520320	0	165152	0	858765	1	913841
	0.000	4447.180	0.000	4028.098	0.000	4797.570	0.002	4290.333
		2.05392E-01		1.85997E-01		2.21546E-01		1.98177E-01
36	0	454716	0	144658	0	755496	1	804629
	0.000	3886.461	0.000	3528.244	0.000	4220.648	0.002	3777.601
		1.79496E-01		1.62916E-01		1.94904E-01		1.74493E-01
37	0	402201	0	127656	0	668545	1	724627
	0.000	3437.615	0.000	3113.561	0.000	3734.888	0.002	3402.005
		1.58766E-01		1.43768E-01		1.72473E-01		1.57144E-01
38	0	353168	0	112436	0	588194	1	644217
	0.000	3018.530	0.000	2742.342	0.000	3286.000	0.002	3024.493
		1.39410E-01		1.26627E-01		1.51743E-01		1.39706E-01
39	0	304048	0	96353	0	508753	1	562761
	0.000	2598.701	0.000	2350.073	0.000	2842.196	0.002	2642.070
		1.20021E-01		1.08514E-01		1.31249E-01		1.22041E-01
40	0	265585	0	84247	0	447046	1	506766
	0.000	2269.957	0.000	2054.805	0.000	2497.464	0.002	2379.183
		1.04838E-01		9.48803E-02		1.15330E-01		1.09898E-01
41	0	226947	0	71791	0	385541	1	428583

	0.000	1939.718	0.000	1751.000	0.000	2153.860	0.002	2012.127
		8.95856E-02		8.08522E-02		9.94626E-02		9.29431E-02
42	0	200379	0	63631	0	340403	1	393368
	0.000	1712.641	0.000	1551.976	0.000	1901.693	0.002	1846.798
		7.90981E-02		7.16623E-02		8.78178E-02		8.53063E-02
43	0	164894	0	51781	0	283341	2	326177
	0.000	1409.350	0.000	1262.951	0.000	1582.911	0.004	1531.347
		6.50907E-02		5.83166E-02		7.30968E-02		7.07352E-02
44	0	145293	0	45447	0	248900	2	297040
	0.000	1241.821	0.000	1108.463	0.000	1390.503	0.004	1394.554
		5.73533E-02		5.11831E-02		6.42117E-02		6.44165E-02
45	0	123301	0	38511	1	212569	3	250390
	0.000	1053.855	0.000	939.293	0.003	1187.536	0.006	1175.540
		4.86721E-02		4.33717E-02		5.48390E-02		5.42999E-02
46	0	104532	0	32554	1	181276	3	223038
	0.000	893.436	0.000	794.000	0.003	1012.715	0.006	1047.127
		4.12632E-02		3.66628E-02		4.67659E-02		4.83683E-02
47	0	90514	0	27849	1	156044	3	195319
	0.000	773.624	0.000	679.244	0.003	871.754	0.006	916.991
		3.57297E-02		3.13640E-02		4.02565E-02		4.23572E-02
48	0	75318	0	23019	1	132279	3	165662
	0.000	643.744	0.000	561.439	0.003	738.989	0.006	777.756
		2.97312E-02		2.59244E-02		3.41256E-02		3.59257E-02
49	0	63107	0	19627	1	110926	4	143634
	0.000	539.376	0.000	478.707	0.003	619.698	0.008	674.338
		2.49110E-02		2.21042E-02		2.86169E-02		3.11487E-02
50	0	53318	0	.16397	1	93191	4	123435
	0.000	455.709	0.000	399.927	0.003	520.620	0.008	579.507
		2.10469E-02		1.84666E-02		2.40416E-02		2.67683E-02
51	0	46371	0	14142	1	82086	7	107015
	0.000	396.333	0.000	344.927	0.003	458.581	0.013	502.418
		1.83046E-02		1.59269E-02		2.11767E-02		2.32074E-02
52	0	38984	0	11710	1	68878	7	92042
	0.000	333.197	0.000	285.610	0.003	384.793	0.013	432.122
		1.53886E-02		1.31880E-02		1.77693E-02		1.99604E-02

53	0	32734	0	9864	1	57879	8	80426
	0.000	279.778	0.000	240.585	0.003	323.346	0.015	377.587
		1.29215E-02		1.11090E-02		1.49317E-02		1.74413E-02
54	0	26557	0	8032	2	48424	9	67433
	0.000	226.983	0.000	195.902	0.006	270.525	0.017	316.587
		1.04832E-02		9.04577E-03		1.24925E-02		1.46236E-02
55	0	22503	0	6820	3	40927	9	57736
	0.000	192.333	0.000	166.341	0.009	228.642	0.017	271.061
		8.88289E-03		7.68079E-03		1.05584E-02		1.25207E-02
56	0	18624	0	5609	3	34059	10	51037
	0.000	159.179	0.000	136.805	0.009	190.274	0.019	239.610
		7.35168E-03		6.31695E-03		8.78661E-03		1.10680E-02
57	0	14980	0	4540	3	27862	13	39928
	0.000	128.034	0.000	110.732	0.009	155.654	0.025	187.455
		5.91324E-03		5.11302E-03		7.18789E-03		8.65884E-03
58	0	13250	0	3991	3	24496	13	36655
	0.000	113.248	0.000	97.341	0.009	136.849	0.025	172.089
		5.23034E-03		4.49473E-03		6.31952E-03		7.94906E-03
59	0	10078	0	3022	3	19093	16	28219
	0.000	86.137	0.000	73.707	0.009	106.665	0.031	132.484
		3.97822E-03		3.40343E-03		4.92565E-03		6.11961E-03
60	0	8758	0	2612	3	16529	17	25632
	0.000	74.855	0.000	63.707	0.009	92.341	0.032	120.338
		3.45715E-03		2.94168E-03		4.26418E-03		5.55859E-03
61	0	7008	0	2077	4	13503	17	20640
	0.000	59.897	0.000	50.659	0.013	75.436	0.032	96.901
		2.76636E-03		2.33915E-03		3.48353E-03		4.47602E-03
62	0	5789	0	1706	4	11383	18	17891
	0.000	49.479	0.000	41.610	0.013	63.592	0.034	83.995
		2.28516E-03		1.92132E-03		2.93661E-03		3.87987E-03
63	0	4893	0	1390	4	9397	19	15379
	0.000	41.821	0.000	33.902	0.013	52.497	0.036	72.202
		1.93148E-03		1.56544E-03		2.42426E-03		3.33511E-03
64	0	3900	0	1077	5	7655	23	12265

	0.000	33.333	0.000	26.268	0.016	42.765	0.044	57.582
		1.53950E-03		1.21293E-03		1.97485E-03		2.65981E-03
65	0	3195	0	868	6	6335	25	10510
	0.000	27.308	0.000	21.171	0.019	35.391	0.048	49.343
		1.26120E-03		9.77556E-04		1.63432E-03		2.27921E-03
66	0	2571	0	708	6	5132	29	8643
	0.000	21.974	0.000	17.268	0.019	28.670	0.055	40.577
		1.01488E-03		7.97361E-04		1.32396E-03		1.87433E-03
67	0	2137	0	603	7	4344	33	7452
	0.000	18.265	0.000	14.707	0.022	24.268	0.063	34.986
		8.43565E-04		6.79108E-04		1.12067E-03		1.61605E-03
68	0	1664	0	455	7	3458	42	5796
	0.000	14.222	0.000	11.098	0.022	19.318	0.080	27.211
		6.56852E-04		5.12428E-04		8.92101E-04		1.25693E-03
69	0	1374	0	365	7	2818	47	4911
	0.000	11.744	0.000	8.902	0.022	15.743	0.090	23.056
		5.42376E-04		4.11069E-04		7.26993E-04		1.06501E-03
70	0	1095	0	311	8	2327	55	3948
	0.000	9.359	0.000	7.585	0.025	13.000	0.105	18.535
		4.32243E-04		3.50253E-04		6.00324E-04		8.56169E-04
71	0	908	0	262	9	1895	59	3417
	0.000	7.761	0.000	6.390	0.028	10.587	0.113	16.042
		3.58426E-04		2.95069E-04		4.88876E-04		7.41015E-04
72	0	713	0	205	12	1576	70	2656
	0.000	6.094	0.000	5.000	0.038	8.804	0.134	12.469
		2.81451E-04		2.30874E-04		4.06579E-04		5.75984E-04
73	0	552	0	161	13	1251	76	2083
	0.000	4.718	0.000	3.927	0.041	6.989	0.145	9.779
		2.17898E-04		1.81321E-04		3.22735E-04		4.51722E-04
74	1	419	0	148	13	1037	80	1869
	0.009	3.581	0.000	3.610	0.041	5.793	0.153	8.775
		1.65397E-04		1.66680E-04		2.67527E-04		4.05314E-04
75	1	344	0	122	13	860	84	1483
	0.009	2.940	0.000	2.976	0.041	4.804	0.160	6.962
		1.35791E-04		1.37398E-04		2.21864E-04		3.21605E-04

76	1	260	1	92	16	662	86	1208
	0.009	2.222	0.023	2.244	0.050	3.698	0.164	5.671
		1.02633E-04		1.03612E-04		1.70784E-04		2.61969E-04
77	2	214	2	72	17	518	100	916
	0.017	1.829	0.045	1.756	0.053	2.894	0.191	4.300
		8.44749E-05		8.10876E-05		1.33635E-04		1.98645E-04
78	2	172	2	58	20	407	107	748
	0.017	1.470	0.045	1.415	0.063	2.274	0.204	3.512
		6.78957E-05		6.53205E-05		1.04999E-04		1.62212E-04
79	2	145	2	47	20	323	117	596
	0.017	1.239	0.045	1.146	0.063	1.804	0.223	2.798
		5.72377E-05		5.29322E-05		8.33282E-05		1.29249E-04
80	3	96	2	40	22	242	138	456
	0.026	0.821	0.045	0.976	0.069	1.352	0.263	2.141
		3.78953E-05		4.50486E-05		6.24316E-05		9.88888E-05
81	3	80	2	35	25	200	145	395
	0.026	0.684	0.045	0.854	0.078	1.117	0.277	1.854
		3.15794E-05		3.94176E-05		5.15964E-05		8.56603E-05
82	5	58	2	25	26	155	161	300
	0.043	0.496	0.045	0.610	0.081	0.866	0.307	1.408
		2.28951E-05		2.81554E-05		3.99872E-05		6.50584E-05
83	5	38	3	23	30	120	190	214
	0.043	0.325	0.068	0.561	0.094	0.670	0.363	1.005
		1.50002E-05		2.59030E-05		3.09578E-05		4.64083E-05
84	6	30	3	18	33	92	202	175
	0.051	0.256	0.068	0.439	0.103	0.514	0.385	0.822
		1.18423E-05		2.02719E-05		2.37343E-05		3.79507E-05
85	9	20	6	11	36	53	217	113
	0.077	0.171	0.136	0.268	0.112	0.296	0.414	0.531
		7.89485E-06		1.23884E-05		1.36730E-05		2.45053E-05
86	12	18	8	10	43	41	225	108
	0.103	0.154	0.182	0.244	0.134	0.229	0.429	0.507
		7.10537E-06		1.12622E-05		1.05773E-05		2.34210E-05
87	16	13	9	5	54	26	244	68

	0.137	0.111	0.205	0.122	0.169	0.145	0.466	0.319
		5.13165E-06		5.63108E-06		6.70753E-06		1.47466E-05
88	22	9	10	5	64	22	267	47
	0.188	0.077	0.227	0.122	0.200	0.123	0.510	0.221
		3.55268E-06		5.63108E-06		5.67560E-06		1.01925E-05
89	25	8	11	2	73	11	301	31
	0.214	0.068	0.250	0.049	0.228	0.061	0.574	0.146
		3.15794E-06		2.25243E-06		2.83780E-06		6.72270E-06
90	31	5	17	2	91	6	330	22
	0.265	0.043	0.386	0.049	0.284	0.034	0.630	0.103
		1.97371E-06		2.25243E-06		1.54789E-06		4.77095E-06
91	34	4	17	2	108	5	347	18
	0.291	0.034	0.386	0.049	0.338	0.028	0.662	0.085
		1.57897E-06		2.25243E-06		1.28991E-06		3.90351E-06
92	42	3	23	1	131	4	391	12
	0.359	0.026	0.523	0.024	0.409	0.022	0.746	0.056
		1.18423E-06		1.12622E-06		1.03193E-06		2.60234E-06
93	56	1	26	1	160	2	436	5
	0.479	0.009	0.591	0.024	0.500	0.011	0.832	0.023
		3.94743E-07		1.12622E-06		5.15964E-07		1.08431E-06
94	71	1	29	1	193	0	459	3
	0.607	0.009	0.659	0.024	0.603	0.000	0.876	0.014
		3.94743E-07		1.12622E-06		0.00000E+00		6.50584E-07
95	90	1	39	1	245	0	478	2
	0.769	0.009	0.886	0.024	0.766	0.000	0.912	0.009
		3.94743E-07		1.12622E-06		0.00000E+00		4.33723E-07
96	104	0	41	0	296	0	500	0
	0.889	0.000	0.932	0.000	0.925	0.000	0.954	0.000
		0.00000E+00		0.00000E+00		0.00000E+00		0.00000E+00

Appendix E. Examples of Errors by HN7

The following are instances of type I and type II error committed by HN7 when tested with the Nesbit word data. Correct spellings appear in the left column, the corresponding misspellings appear in the middle column, and the value returned by HN7 for each word-misspelling pair appears in the right column. All type I errors are given here, but only a small randomly selected portion of the total number of type II errors are presented.

Type I Error

accept/ed	aspect	56
ache/	aaek	48
	ack	66
	eak	31
actor/	aktter	67
already/	olrede	64
alway/s	awes	66
among/	amuge	68
any/	enoy	62
arriv/ed	drive	58
author/	awither	74
away/	uay	59
beggar/	pegger	71
cedar/	cittar	65
	seader	72
	seator	56
	seatter	52
	sedder	63
	seder	71
	seeder	67
	seedor	67
	setar	66
certain/	centen	70
cinderella/	sindrelue	70
eighty/	etei	44
engage/	agage	72

	ingade	69
geolog/y	gedgily	58
grammar/	crammer	73
ladder/	latter	73
major/	mager	71
	manger	63
massag/e	masach	68
	masosh	68
	mesash	68
	musoshe	67
	musouge	71
mayor/	maiar	71
	maire	68
	mangor	72
	marrior	67
	marrir	63
	marror	72
mirror/	mere	56
	mieeor	69
mutton/	meten	70
	moten	70
ninety/	nighty	69
ocean/	otion	61
pas/sed	paste	72
pimples/	pimppal	69
prince/	pries	74
raze/	raies	67
	rais	58
	raise	69
	rays	55
read/y	rede	69
	redte	63
right/	write	42
rough/	rofe	53
	roff	50
	ruff	50

ruf/f	rouf	71
	roufe	69
	rough	50
simpl/e	simpaill	73
size/	sies	71
sour/	sawr	65
	sror	71
squ/are	saer	73
successiv/e	secseveve	69
	sicseciv	74
	sicsesof	65
	succesof	74
	succeufe	71
	suckseof	64
	sucseccof	65
	sucsesof	71
term/s	turns	71
through/	thour	67
towel/	taule	66
wrist/	rist	72
writ/ing	righting	67

Type II Error

ache/	ace	86
	achmet	76
	acre	77
	apache	78
	arched	76
	archer	76
	arches	76
	ash	76
	ashes	75
	cache	81
anything/	nothing	78
bought/	blight	76
	bough	88
	boughs	84
	bright	76
	brought	90
	fought	80
	nought	78
	ought	77
	sought	78
cattle/	battle	78
	cale	76
	castle	84
	castles	76
	rattle	78
	seattle	80
	settle	76
chemi/stry	cemetery	75
geolog/y	ecology	78
	gloomy	76
	theology	75
instrument/	installment	77
	instant	75
	investment	77
measle/s	males	83
	maples	80
	mass	76
	masses	80
	meals	83
	measures	81
	medals	76
	mesas	79
	messages	75
	metals	76

	miles	75
	miracles	77
	moles	75
	moses	75
	moslems	77
	mules	75
	muscles	77
	mussels	75
	weasels	77
possib/le	impossible	80
	passable	86
	permissible	77
puzzl/e	muzzle	78
rough/	rug	76
	trough	77
slic/e	silence	79
	silica	81
	since	76
	slide	82
	slime	82
	solace	81
	spice	82
writ/ing	awaiting	75
	rewriting	79
	waiting	87
	wanting	75
	warming	75
	warning	75
	warring	75
	wasting	75
	wetting	75
	whirling	77
	whirring	77
	wiping	76
	wiring	86
	wising	76
	wording	77
	working	75
	wrestling	77
	wring	81
	wringing	79

SPECIAL COLLECTIONS

UNIVERSITY OF ALBERTA LIBRARY

REQUEST FOR DUPLICATION

I wish a photocopy of the thesis by

John Cale Nesbit (author)

entitled Approximate String Matching Applied to Response Analysis in Computer Assisted Instruction

The copy is for the sole purpose of private scholarly or scientific study and research. I will not reproduce, sell or distribute the copy I request, and I will not copy any substantial part of it in my own work without permission of the copyright owner. I understand that the Library performs the service of copying at my request, and I assume all copyright responsibility for the item requested.